SONDERHEFT 52 Markt&Technik ÖS 120,-/Sfr. 16,-/Lit. 16000 hfl. 21,-/dkr. 75,-/fmk 43,- DM 16,-

DISKETTE IM HEFT

64'er

ABENTEUERSPIELE

Spannend

8 Rollen- und Fantasyspiele auf Diskette

Selbst programmieren

Von der Idee zum fertigen Spiel

Entschlüsselt

So knacken Sie Adventures

Neu überarbeitete Auflage



DISKETTE IM HEFT 64'er

POWER-GAMES

Jedes Paket nur DM 49,–* (sFr 45,–/*öS 490,–*)

ACTION • SPANNUNG • ABENTEUER

Das Schwert Skar
Skar verleiht seinem Träger elementare Kräfte. Es macht ihn unbesiegbar und unsterblich. Aber es ist gut versteckt! Wer es finden will, muß den Gefahren eines langen Weges trotzen.
Bestell-Nr. 38784

Die Flucht der Sumpfgeister
Als die Menschen begannen, die Sümpfe trocken zu legen, haben die Sumpfgeister mit dem einzigen vorhandenen magischen Staubsauger die Flucht zu einem weit entfernten Planeten ergriffen. Ein Sumpfgeist hat jedoch die Abreise verschlafen...
Bestell-Nr. 38785

POWER-GAMES erhalten Sie im guten Fachhandel

Operation Ushkurat
Sie sind mit einem Raumschiff unterwegs zu Friedensverhandlungen. Bei einer Reparatur wird die gesamte Mannschaft entführt...
Bestell-Nr. 38765

Dungeon
»Dungeon« ist eine Variante des legendären Spieleklassikers »PacMan«. Die Spielfigur bewegt sich durch ein Labyrinth. Eingebaute Türen, Teleporter sowie diverse Hilfsmittel helfen Ihnen, Geistern und Monstern aus dem Weg zu gehen...
Bestell-Nr. 38760

Operation Feuersturm
Sie sind »Mister James Bond« und haben 48 Stunden Zeit, eine gestohlene Atombombe zu finden – falls nicht, wird sie abgefeuert.
Bestell-Nr. 38739

Howard the Coder
Howard hat eine Spielidee. Leider stiehlt man seinen Computer und er sucht sich in einer Lagerhalle neue Hardware zusammen. Dabei muß er Hindernisse überwinden...
Bestell.-Nr. 38705

Mit Jeans und Hellebarde
Bei der Reparatur eines Schuppens stürzt die Decke herab und macht Sie kampfunfähig. Als Sie zu sich kommen entdecken Sie ein altes Buch mit merkwürdigen Buchstabenkombinationen. Sie wissen noch nicht, daß Sie Ihre Welt bereits verlassen haben...
Bestell-Nr. 38718

Nippon – das ultimative Rollenspiel für C 64/C 128
Toshiro begann, die zufällig entdeckten Schriftrollen zu lesen. Sie sahen abgegriffen und uralt aus... Vor Ihnen liegt ein Abenteuer, wie Sie es bisher nicht gekannt haben.
Bestell-Nr. 38729

* Unverbindliche Preisempfehlung.



Markt&Technik
Zeitschriften · Bücher
Software · Schulung

2310/912

INFO-COUPON
Bitte senden Sie mir Ihr Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software.

☐ Ich bin Fachhändler

Name ______________________

Straße ______________________

PLZ/Ort ______________________

Bitte ausschneiden und senden an: Markt&Technik Verlag AG, Buch- und Software-Verlag, Hans-Pinsel-Str. 2, 8013 Haar bei München

64 SH 52

64'er SONDERHEFT 52

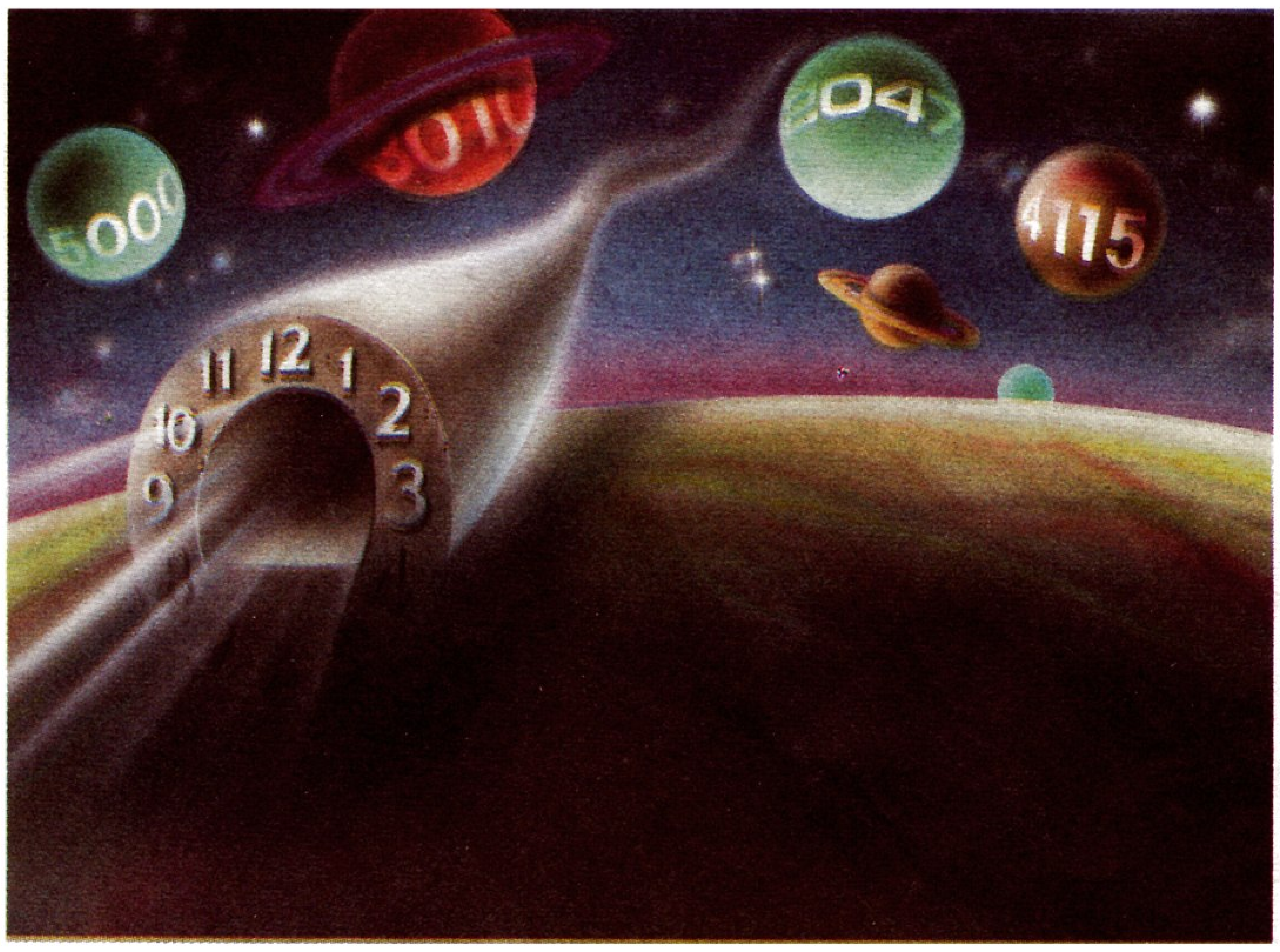
Abenteuerspiele

Tips & Tricks

Grundlagen



Welches ist der richtige Weg? Reißende Flüsse, steile Berge und feindliche Krieger sind zu überwinden.
Seite 12



Wunschtraum aller Science-fiction-Fans: eine Reise durch den Zeittunnel. Kein Weg scheint aus der Zukunft zurückzuführen.
Seite 8



Adventures programmieren kann jeder. Mit unserem ausführlichen Kurs wollen wir diese Behauptung beweisen.
Seite 16

Sonstiges

Alle Programme aus Artikeln mit einem ▪-Symbol finden Sie auf der beiliegenden Diskette (Seite 35).

GEHEIM-AUFTRAG FÜR 008

Das Ultimatum läuft. Nur ein paar Stunden bleiben Ihnen, um die Atomrakete zu finden, die den dritten Weltkrieg auslösen soll. Ein gefährlicher Auftrag führt Sie ins Reich der aufgehenden Sonne.

James Bond liegt mit Grippe im Bett. Der amerikanische Präsident wählt deshalb Sie für den geheimsten Auftrag aller Zeiten aus: Ein hoher Offizier der Luftwaffe hat eine Rakete mit Atomsprengkopf gestohlen und nach China gebracht. Der Erpresser droht, den dritten Weltkrieg auszulösen, wenn man nicht auf seine Bedingungen eingeht. Um seine Forderung noch drastischer zu unterstreichen, hat er die Atomrakete aktiviert – der Countdown läuft. Ihnen bleibt nicht viel Zeit, das Versteck aufzuspüren, die Rakete zu entschärfen und die Menschheit vor dem Untergang zu retten.

Um keine Zeit zu verlieren, laden Sie das Spiel mit

```
LOAD "ADVENTURE 2000",8
```

und starten es mit RUN. Die kurze Maschinensprache-Routine »SCROLL.MC« wird automatisch nachgeladen.

Das Adventure führt Sie zunächst in die Amtsräume des Präsidenten (Bild 1). Er übergibt Ihnen streng vertraulich eine Mappe. Sie öffnen diese und lesen die Anweisungen. Das eigentliche Abenteuer beginnt an der Chinesischen Mauer. Mehr möchten wir nicht verraten.

»Adventure 2000« versteht Eingaben mehrerer Worte. Man kann bestimmte Signal- und Bindewörter zum Objekt eingeben, ohne den Satz allzusehr verstümmeln zu müssen. Ein Beispiel:

```
NIMM DEN SCHLUESSEL DANN STECKE DEN SCHLUESSEL IN
DAS SCHLUESSELLOCH DANN OEFFNE DIE TUERE UND GEHE
DURCH DIE TUERE
```

Dieser Text wird von den Routinen des Spiels rasch bearbeitet.

Die Eingabezeile befindet sich unter dem jeweiligen Grafikbild für den entsprechenden Raum. Die Stelle, an der Sie Befehle eingeben können, ist mit einer spitzen Klammer kenntlich gemacht. Tippen Sie das Wort »BEFEHLE« ein, listet der Computer alle Verben auf, die er versteht:

```
GEH, NIMM, LEG, SCHIESSE, WERFE (WIRF), GEBE
(GIB), STARTE, ZIEHE, OEFFNE, LESE (LIES), WAELZE,
DRUECKE, KLETTERE, RUFE, FUELLE, STECKE,
UNTERSUCHE, TRINKE.
```

Grundsätzlich müssen mindestens zwei Worte eingegeben werden (Verb und Objekt). Fügen Sie Bindewörter oder Artikel (z. B. der, die, das) hinzu, tritt die Befehlserkennungsroutine des Adventures in Aktion. Sie beginnt ab Programmzeile 290 und filtert folgende Eingabewörter aus: der, die, das, den, durch, nach. Satzattribute wie »und« oder »dann« bewirken eine interne Markierung in der Befehlseingabe mit dem <*>-Zeichen. Ähnliches läuft ab, wenn die Codierroutine des Abenteuerspiels auf Wörter wie »mit«, »in«, »auf« oder »an« stößt. Statt des »Sternchens« erscheint jetzt das Plus-Zeichen < + > als markanter Hinweis für den Computer. Erst nach diesem Filtiervorgang geht die Routine zur Interpretation der eigentlichen Befehlserkennungsroutine über. Nun wird die Befehlserkennungsroutine

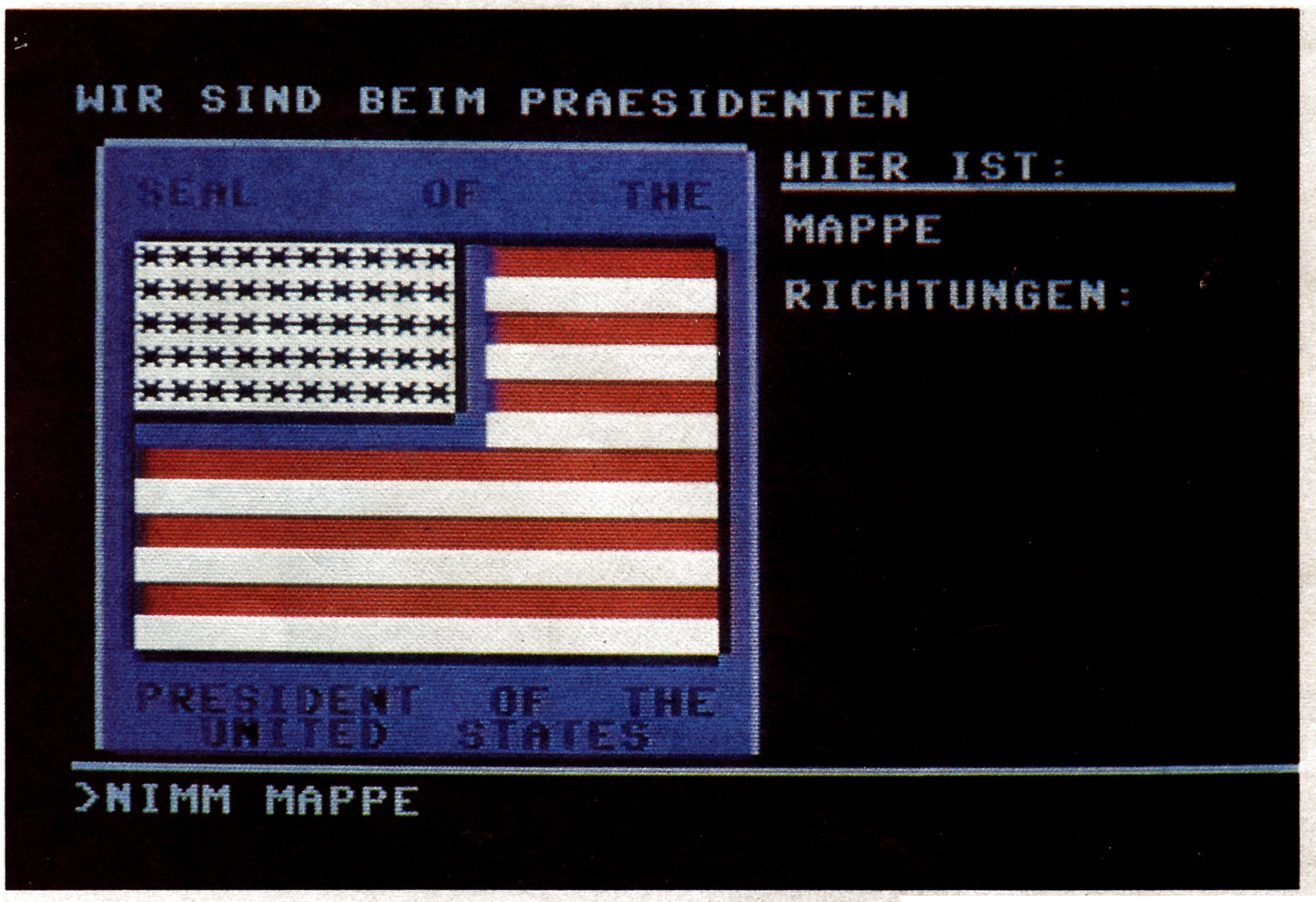


Bild 1. Sie erhalten Ihren Geheimauftrag. Das Abenteuer beginnt.

durchlaufen. Geben Sie falsche Befehle ein, macht Sie der Computer darauf aufmerksam. Bei folgenden Wörtern besteht eine Ausnahme, sie werden vom Programm auch als Ein-Wort-Eingaben akzeptiert:

```
UMSCHAUEN, SIEH, BEFEHLE, STOP, ZUEGE, LIST, TAUCHE
```

Es genügt, von jedem Verb oder Objekt nur die ersten drei Buchstaben anzugeben, den Rest erkennt das Programm automatisch. Signalwörter wie »dann« oder »und« müssen immer ausgeschrieben werden.

Beim Betreten eines Raumes sehen Sie in der obersten Bildschirmzeile, wo Sie sich befinden. Links wird die Grafik des entsprechenden Raumes gezeigt (im Spiel sind 24 Räume enthalten), rechts, welche Objekte im Raum sind und welche direkten Ausgänge (Richtungen) es gibt. Fehlen solche Hinweise, ist das vom Programmautor bewußt so eingerichtet. Es bleibt dem Kombinationsvermögen des Spielers überlassen, andere zu finden.

Der Computer überprüft die Eingabe (sie muß mit der RETURN-Taste abgeschlossen werden). Wurden unbekannte Worte verwendet, teilt er Ihnen mit, daß er nicht weiß, wovon die Rede ist. Erkennt er die Eingabe, erscheint die entsprechende Antwort auf dem Bildschirm.

Dazu ein Beispiel:

```
NIMM MAPPE
OK. GENOMMEN: MAPPE.
```

Sollte der Computer nach langem Spielverlauf Ihre Befehlseingaben nicht mehr verstehen, liegt das am String-Müll, der sich bis dahin angesammelt hat. Mit der Taste <F1> zwingen Sie ihn wieder zur Mitarbeit an der Lösung des Adventures. Wir hoffen, Sie finden die Atomrakete und Major Mc Perkins rechtzeitig. (Frank Schmelzer/bl)

Kurzinfo: Adventure 2000

Programmart: Abenteuerspiel
Spielziel: Finden Sie die entführte Atomrakete und entschärfen Sie sie.
Laden: LOAD "ADVENTURE 2000",8
Starten: Nach dem Laden RUN eingeben
Steuerung: Tastatur
Benötigte Blocks: 138 Blocks
Programmautor: Frank Schmelzer

Basic-Zeilen	Programmfunktion
100	Arrays dimensionieren
120 – 154	Raum wurde betreten
158 – 280	Eingabe (ersetzt INPUT, um verbotene Zeichen zu verhindern)
290 – 380	Unterprogramm Eingabe erkennen und bearbeiten
400 – 510	Erstellen der aktuellen Zwei-Wort-Eingabe, falsche Wörter zurückweisen, aktuelles Verb feststellen
510	springe in den Programmteil, der sich mit aktuellem Verb befaßt
520 – 1620	Zwei- und Mehr-Wort-Kommandos
2000 – 2100	Überprüfe Raum und dessen Objekte, erhöhe Variablen, setze Wachen in Räume etc.
2110 – 2220	Erkenne und ersetze verschiedene Verben
2230 – 2410	Ein-Wort-Kommandos werden abgeschaltet
9000 – 9130	Text »Anweisungen«
9140 – 9190	Sie haben gewonnen!
10000 – 13880	Definition der Raumbilder
12880 – 13911	Einlesen der Daten
13920 – 14170	Daten (Befehlsworte, Objekte, Räume)
14370 – 14400	Spiel verloren

Tabelle 1. Programmübersicht von »Adventure 2000«

DEM HEXER AUF DER SPUR

Niemand hat ihn je zu Gesicht bekommen, seinen Namen wagt kaum jemand auszusprechen: Quasimodo, der Hexenmeister. Nehmen Sie Ihren ganzen Mut zusammen und spüren Sie sein Versteck auf.

In grauer Vorzeit herrschte der grausame Zauberer Quasimodo im Reich der Kartanen. Glücklicherweise machten Abgesandte des unterjochten Volkes einen strahlenden Helden ausfindig, der das Volk von diesem blutrünstigen Tyrannen befreien soll – nämlich Sie. Ihr gesatteltes Pferd (pardon: eingeschalteter C 64) steht bereit, die Kartanen bringen Sie durch einen Geheimgang in Quasimodos Gewölbe. Nur Mut, kühner Recke!

Laden Sie das Grafik-Adventure mit

```
LOAD "QUASIMODO",8
```

Nach dem Einlesen der Daten für die Adventure-Bilder und Sprites befinden Sie sich in einem Raum des unterirdischen Labyrinths, in dem sich der Bösewicht versteckt hält. Zum ersten Mal stockt Ihnen der Atem: Sie stoßen auf ein bedauernswertes Opfer des Tyrannen. Auf diese Weise verfährt er mit allen Widersachern. Das fängt ja gut an...

Das Adventure besteht aus 31 Räumen. Die sich darin befindlichen Gegenstände werden durch farbige, grafisch gut gestaltete Multicolor-Sprites dargestellt. Die Befehls-Routine des Spiels versteht nur Zwei-Wort-Eingaben wie z.B.:

```
NIMM FLASCHE
```

Das Programm stellt beim Start den Großschrift-Modus ein, Kleinbuchstaben können nicht eingetippt werden. Folgende Verben werden im Adventure verwendet:

```
NIMM, GIB, GEH, LISTE, HOCH, ZUENDE, SPRING, TRINK,
DRUECKE, BEFEHLE, OEFFNE
```

LISTE zeigt alle Gegenstände, die Sie gerade besitzen, auf dem Bildschirm an.

Kurzinfo: Quasimodo

Programmart: Abenteuerspiel
Spielziel: Den bösen Zauberer aufspüren und eliminieren
Laden: LOAD "QUASIMODO",8
Starten: Nach dem Laden RUN eingeben
Steuerung: Tastatur
Benötigte Blocks: 50 Blocks
Programmautor: Roland Selzer

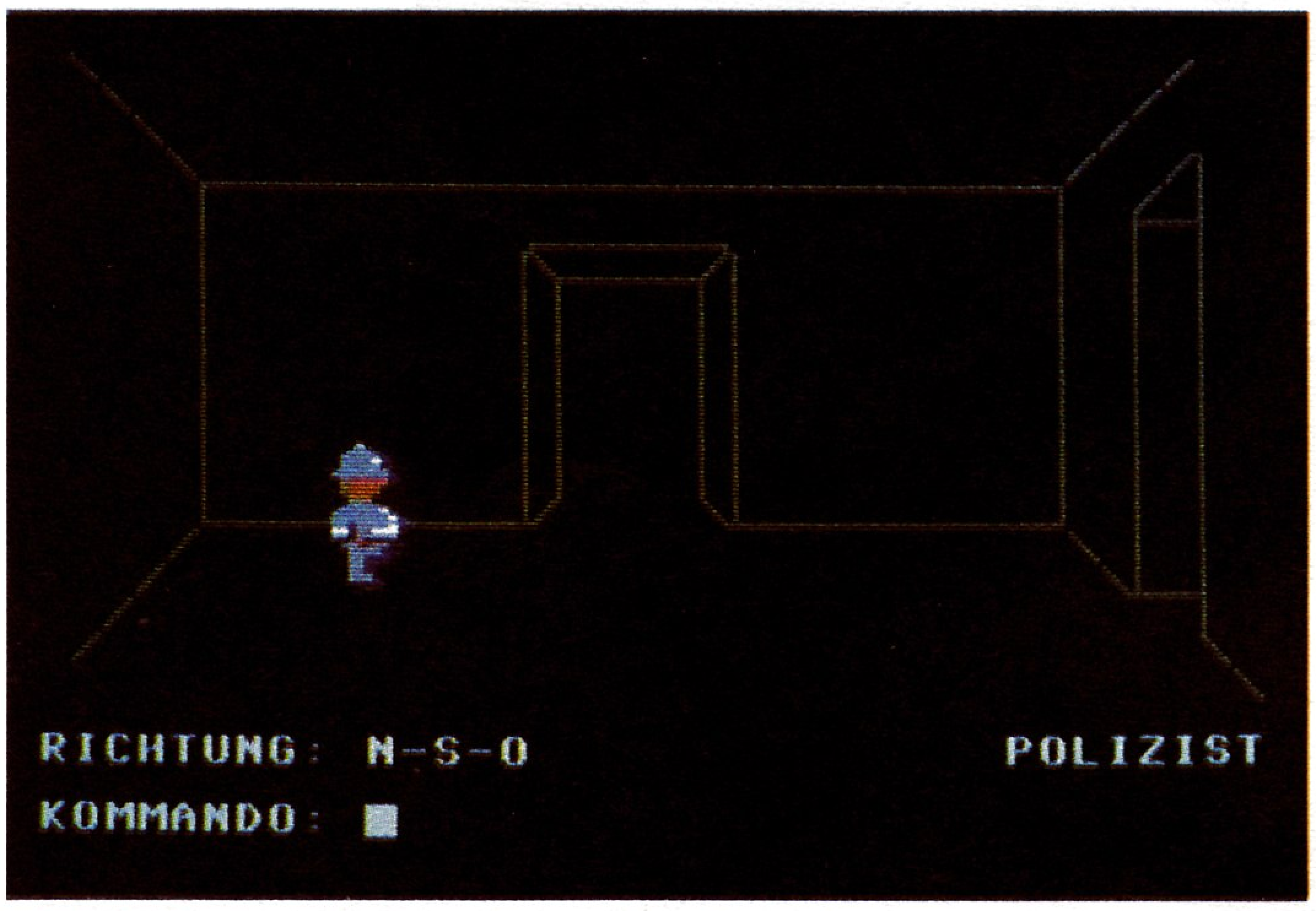


Bild 1. Das wachsame Auge des Gesetzes

Alle erkennbaren Verben kann man sich unmöglich merken.

Befehle aktiviert eine Laufschrift, die in lesbarer Geschwindigkeit die erlaubten Befehlsverben am Bildschirm vorüberziehen läßt.

Richtungsangaben müssen auf einen Buchstaben reduziert werden (z.B. »N« für Norden usw.). Welche Richtungen im jeweiligen Raum möglich sind, zeigt des Programm oberhalb der Befehlseingabezeile. Die Objektbezeichnungen sind rechts angegeben.

Tips zum Spiel

Da das Adventure eine große Anzahl an Räumen besitzt, kann man sich leicht verlaufen. Am besten notieren Sie sich die jeweils letzten zwei Richtungsänderungen. Geben Sie vor allen Dingen acht, ob und was Sie im entsprechenden Raum an sich nehmen. Eine Zimmerpflanze ist zwar ein erfreulicher Anblick, aber sie zu berühren, kann seine Tücken haben. Oder: Spielen Sie nicht mit dem Feuer, das ist gefährlich. Und Polizisten sollte man nicht reizen (Bild 1). Entscheidende Spielsituationen werden vom Programm akustisch untermalt.

Untersuchen Sie nur solche Gegenstände, die Ihnen wichtig erscheinen. Halten Sie in den verschiedenen Räumen des Adventures vor allem nach Waffen oder ähnlichen Dingen Ausschau, mit denen Sie sich verteidigen können. Dem Tyrannen schutzlos gegenüberzutreten, wäre reiner Selbstmord. Sollten Sie auf Dynamit stoßen, so denken Sie daran, daß man es nur mit einem Streichholz oder einem Feuerzeug zünden kann. Durch den entsprechenden Befehl können Sie nach oben springen. Dabei hat sich so mancher den Kopf kräftig angestoßen. Noch ein letzter Tip: In einer Flasche muß sich nicht unbedingt Gift befinden.

Alles klar? Nach diesen »ausführlichen« Hinweisen wollen wir Sie mit dem Hexenmeister allein lassen. Spüren Sie ihn auf und vernichten Sie ihn. (Roland Selzer/bl)

Zeile	Programmfunktion
20 – 670	Titelbild und Variablen
700 – 715	Einlesen der Sprite-Daten
875 – 1092	Aufbau der Raumgrafik
1093 – 1094	Unterprogramme für Sprites aufrufen
1100 – 1230	Befehlseingabe
1240 – 3002	Eingaben verarbeiten
3005 – 3299	Aufbau der Türengrafik
3310 – 3975	Sprite-Daten (22 Sprites)
4000 – 4950	Reaktion auf Kommandos
5000 – 5970	Unterprogramme Sprites
6000 – 6060	Laufschrift
6500 – 6520	Töne
6540 – 9999	Unterprogramme Sprites

Tabelle 1. Der Programmaufbau von »Quasimodo«

UN

Sie sind Dorfo, der Hobbit, und verkörpern das »Gute«. Bei Ihrem abenteuerlichen Vorhaben, die Welt vor dem »Bösen« zu retten, haben Sie schon die vielfältigsten Gefahren überwunden. Nun stehen Sie nach langer Wanderung vor der verlassenen Mine »Mario« (Bild 1). Durchqueren Sie dieses von grausigen Wesen bewohnte Labyrinth ohne Schaden zu erleiden (Bild 2). »Dangalf«, ein alter und weiser Zauberer, steht Ihnen dabei zur Seite.

»Mario« ist der Zwergenmine »Moria« aus dem Werk »Der Herr der Ringe« von J.R. Tolkien nachempfunden. Sollten Sie Schwierigkeiten bei der Lösung dieses Grafik-Adventures haben, empfiehlt es sich, in diesen Büchern nachzulesen.

Das Programm ist zum größten Teil in Basic geschrieben. Eine Maschinenroutine wird zu Beginn des Spiels nachgeladen. Durch den ersten Teil dieses Maschinenspracheprogrammes werden die Mängel der GET-Routine des Betriebssystems beseitigt:
- Zugelassen sind alle Buchstaben ohne <SHIFT>
- Alle Ziffern sind als Eingabe erlaubt
- Sonderzeichen sind ebenfalls erlaubt. Ausnahme: <">, <,> und <:>
- <Shift CLR/HOME> löscht nur die Eingabezeile
- <CLR/HOME> springt an den Anfang der Eingabezeile
- <INST> funktioniert wie gewohnt
- <DEL> fügt ein Leerzeichen unter dem Cursor ein und verschiebt um ein Zeichen nach rechts. Das letzte Zeichen der Zeile geht verloren. Eine anschließende Cursorbewegung wird korrekt durchgeführt.
- Die Cursorbewegungen links und rechts bleiben wie gewohnt erhalten
- Cursor nach oben und nach unten sind gesperrt.

Die Eingaberoutine läßt sich für eigene Spieleprogrammierung verwenden. Sie ist auf der Diskette unter »SCROLL/1« gespeichert. Verwendet wird Sie beispielsweise so:

```
100 PRINT : SYS 49152 + 256: INPUT A$
```

Wobei die Zeilennummer (100) beliebig sein kann und der String (A$) natürlich jede andere erlaubte Bezeichnung besitzen kann.

Der zweite Teil dieses Maschinenprogramms läßt im Gegensatz zum Betriebssystem nur die letzten sechs Bildschirmzeilen scrollen und setzt die Farbe der gescrollten Zeichen auf Schwarz. Bei der eigenen Verwendung ist zu beachten, daß PRINT-Anweisungen nicht länger als 39 Zeichen sein dürfen. Sie starten diese Routine am Anfang Ihres Pro-

Eingaben	Bedeutung
N, S, O, W	Himmelsrichtungen
H	eine Ebene hoch
R	eine Ebene runter
SAGE	wird von beliebigen Worten gefolgt (ohne Anführungszeichen)
HILF	Hilfen oder Erklärungen (mit Vorsicht zu genießen)
LIST	zeigt die getragenen Gegenstände auf (max. drei)
Befehle, die erst nach Betreten der Mine eingegeben werden können	
NIMM, VERLIERE, HEBE, ENTLEERE, WIRF, LIES	verlangen zusätzlich die Eingabe eines Gegenstandes
OEFFNE	eine Tür oder ein Tor
BRINGE ... UM	... muß ein Lebewesen sein
SPEICHERE	den momentanen Spielstand
LADE	einen gespeicherten Spielstand

Tabelle 1. Der Wortschatz von »Mario«

DIE HEIMLICHE MINE

Finstere Gänge, schreckliche Feuerwesen und eine grauenhafte Riesenspinne behindern Sie auf dem Weg durch das unterirdische Labyrinth. Durchforsten Sie die unheimliche Mine.

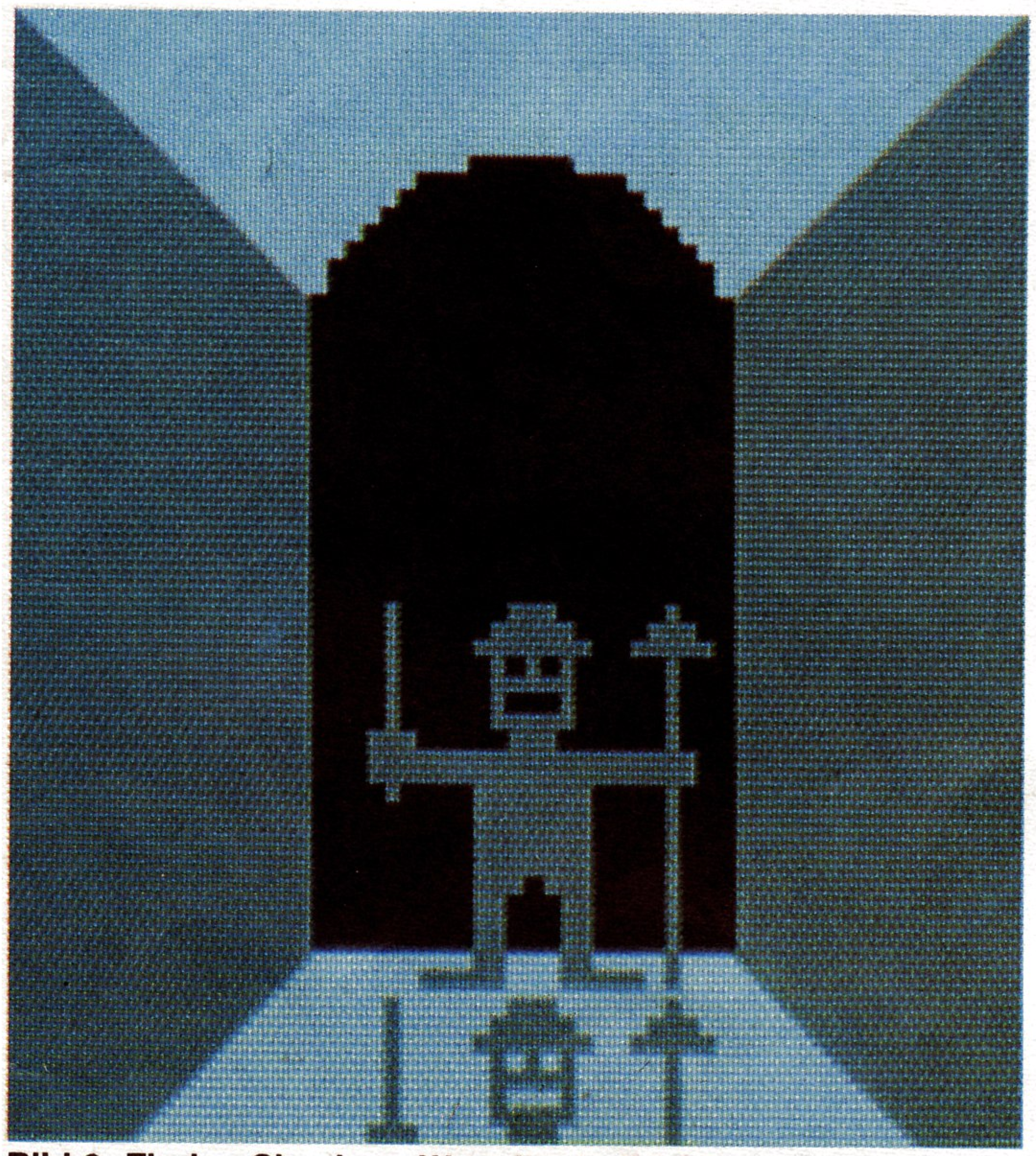


Bild 2. Finden Sie einen Weg durch die Gänge

Kurzinfo: Mario

Programmart: Grafik-Abenteuerspiel
Spielziel: Durchwandern Sie das Labyrinth der Mine, ohne Schaden zu nehmen
Laden: LOAD″MARIO″,8
Starten: Nach dem Laden RUN eingeben
Steuerung: Tastatur
Benötigte Blocks: 114 Blocks Basic-Programm und 3 Blocks Maschinensprache
Programmautor: Harald Bornfleth

gramms mit:

```
SYS 49152
```

Ausgeschaltet wird Sie durch <RUN/STOP RESTORE>.

Beachten Sie die Himmelsrichtungen im Spiel. Osten liegt immer in Blickrichtung. Den Wortschatz von »Mario« entnehmen Sie bitte der Tabelle 1. Die Programmierer unter unseren Lesern finden eine Liste der verwendeten Variablen und ihrer Belegung in Tabelle 2.

Sie laden das Programm von der beiliegenden Diskette mit:

```
LOAD"MARIO",8
```

und starten mit RUN. Zunächst wird das Maschinenprogramm nachgeladen, danach benötigt die Generierung der Sprites ca. 30 Sekunden. Während dieser Zeit haben Sie Gelegenheit, eine Anleitung auf dem Bildschirm zu lesen. (Harald Bornfleth/gr)

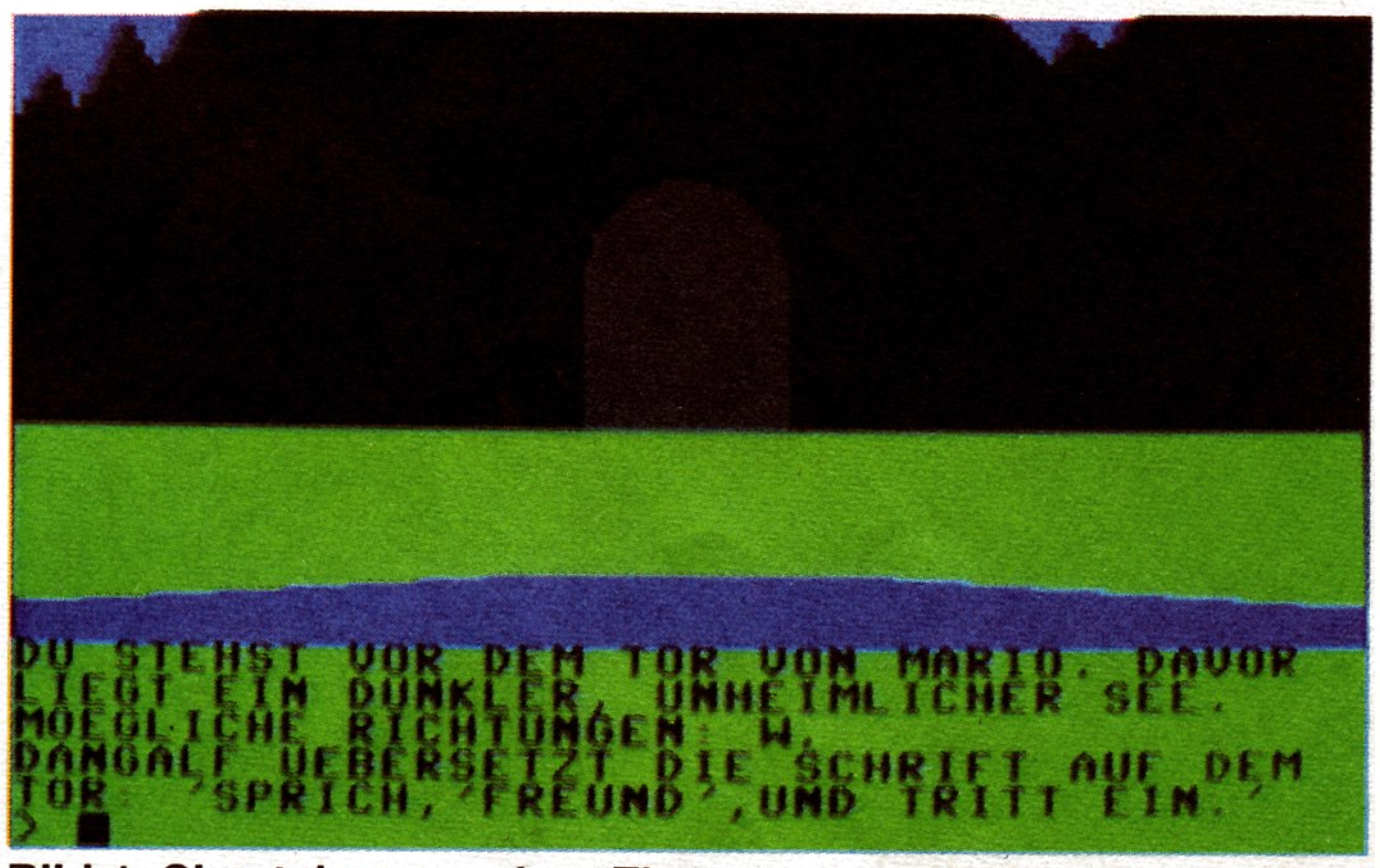


Bild 1. Sie stehen vor dem Eingang von »Mario«

Numerische Variablen

A (23,63)	– Werte der Sprites
B (6)	– Register für Ausgänge aus Räumen
C (6,12)	– Gegenstände in den einzelnen Räumen
D (6,12)	– Ausgänge aus den einzelnen Räumen
E (6,12)	– Besonderheiten in den einzelnen Räumen
HV	– Hilfsvariable
HW	– Hilfsvariable
L1	– Laufvariable
LW	– Laufvariable
LV	– Laufvariable
M	– X-Position Spinne (Zufallszahl)
M1	– Register für gehobene Laterne
M2	– Register für Erschöpfung
M3	– Register für mehrmaliges Betreten des 1. Raumes
M4	– Register für Wasser im Eimer
M5	– Register für Raum mit Feuerabgrund
M6	– Register für geworfenes Seil
M7	– Zahl der absolvierten Spiele
N	– Y-Position Spinne (Zufallszahl)
TU	– Register für offenes/geschlossenes Anfangs/Endtor
X	– Position Spieler
Y	– Position Spieler
String-Variablen	
A$	– Zeichen
A$ (10)	– Gegenstände
B$	– Befehl
C$	– Filename
FG$	– Besonderheiten in einem Raum
GD$	– Eingegebener Gegenstand (zum Beispiel hinter NIMM)
GE$	– Gegenstände in einem Raum
GS$ (10)	– Gegenstände, die man bei sich trägt
RI$	– Richtungen

Tabelle 2. Alle Variablen auf einen Blick



Zurück aus der Zukunft

Etwas Unbegreifliches ist geschehen: Sie sind in den Zeitsog geraten, der Sie in die Zukunft auf einen unbekannten Planeten geschleudert hat. Nur durch das andere Ende des Zeittunnels können Sie wieder in die Gegenwart zurückkehren.

Samstagabend, 20.12 Uhr. Im Fernsehen läuft die »Tagesschau«. Ein seltsamer Lichtschein fällt Ihnen auf, der aus den Ritzen des Wohnzimmerschranks zu kommen scheint. Neugierig öffnen Sie den Schrank. Gleißende Helligkeit umgibt Sie. Sie fühlen sich wie bei einem Orkan herumgewirbelt, das Geschehen von Jahrtausenden zieht in Sekundenschnelle an Ihnen vorbei. Bizarre Figuren, die nicht von dieser Welt sind, begleiten Sie auf Ihrem schwerelosen Weg durch die Zeit. Plötzlich ist alles ruhig. Sie finden sich auf einem kalten Planeten aus Kristall wieder, ohne einen blassen Schimmer, wo Sie sind.



Die geschilderte Anfangssituation ist der Auftakt zu einem spannenden Adventure mit elf farbigen Blockgrafikbildern. Sie laden das Programm mit

```
LOAD "ZEITTUNNEL",8
```

und starten es mit RUN. Das erste Bild zeigt Ihnen die Richtungen und Befehlswörter, die das Spiel versteht.

Mögliche Richtungen:

OB (oben), RU (runter), N (Norden), S (Süden), W (Westen), O (Osten). Diese Angaben dürfen nur mit den genannten Kürzeln erfolgen.

Befehlsworte:

```
NEHME, DRUECKE, GEBE, ZERSTOERE, GEHE DURCH,
BEFRAGE, OEFFNE, TOETE, SAGE.
```

Das Adventure akzeptiert nur Zwei-Wort-Eingaben: Ein Verb und ein Objekt.

Falls Sie statt »OEFFNE SCHRANK« zum Objekt-Substantiv den Artikel verwenden (z.B. OEFFNE **DEN** SCHRANK), wird das als Fehleingabe interpretiert.

Die Grafik für die einzelnen Adventure-Räume erstreckt sich über die gesamte Bildschirmfläche. Das Programm teilt durch die Meldung »Standort« mit, wo Sie sich gerade befinden. Darunter werden die Objekte im Raum angezeigt. Da für Befehlseingaben kein Platz mehr vorhanden ist, benutzt das Spiel dazu einen eigenen Bildschirm, den Sie nach Drücken der Taste <RETURN> aufrufen können. Der obere Bereich des aktivierten Monitor-Bildes gibt eine Kurzbeschreibung zum Raum und zur Spielsituation aus. Um sich die aktuelle Raumgrafik wieder anzeigen zu lassen, müssen Sie die Anweisung »BILD« eingeben.

Tips zum Spiel

Irgendwo im Adventure sind Karten mit Zahlencodes und ein Paßwort versteckt. Versuchen Sie auf alle Fälle, diese Informationen zu finden und notieren Sie sich alle Zahlen und Buchstaben. Sie werden diese Codes gut gebrauchen können, wenn Sie den Transmitter, das andere Ende des Zeittunnels, finden.

Die Codezahlen werden zu Beginn des Spiels durch die RND-Funktion (Zufallsgenerator) erzeugt und gelten nur für die jeweilge Spielrunde. Eine SAVE- und LOAD-Funktion (zum Speichern und Laden des aktuellen Spielstands nach einer Unterbrechung) ist vom Programm nicht vorgesehen.

Trotzdem sind wir überzeugt, daß Sie den Weg zurück aus der Zukunft finden werden.

(Jens Schürks, Arndt Meiswinkel/bl)

Kurzinfo: Zeittunnel

Programmart: Abenteuerspiel
Spielziel: Suchen und finden Sie den Eingang des Zeit-Transmitters, der Sie in die Gegenwart zurückbringt
Laden: LOAD "ZEITTUNNEL",8
Starten: Nach dem Laden RUN eingeben
Steuerung: Tastatur
Benötigte Blocks: 108 Blocks
Programmautor: Jens Schürks/Arndt Meiswinkel

Grafik: Ewald Standke

DIE GEHEIMNISVOL ZENTRALE

Sprengen Sie die Raumstation des Uranminenbesitzers »Boghart«. Finden Sie sein geheimes Hauptquartier und retten Sie eine gesamte Galaxis vor der Zerstörung.

Im weiten All, in der Nähe des Andromeda-Nebels, kreist heimtückisch und leise die Basis des abtrünnigen »Boghart«. Da der imperiale Weltenrat beschlossen hat, allen Trilliardären die Steuern zu erhöhen, beschloß der geldsüchtige Boghart die Herrschaft zu übernehmen.

Der umsichtige Weltenrat kann das auf keinen Fall zulassen und schickte schon Agenten los, die Basis dieses Verbrechers zu zerstören. Leider gelang es bis jetzt keinem, die Mission zu erfüllen. Nun sind Sie, der fähigste Sonderagent, beauftragt, die Galaxis zu retten.

Sie laden das Spiel von der beiliegenden Diskette mit

```
LOAD"FUTURE",8
```

und starten es mit <RUN>.

Nach einer kurzen Wartezeit befinden Sie sich im ersten von 27 Räumen. In diesen sind verschiedene Gegenstände verteilt, die Sie zur Erfüllung Ihrer Mission benötigen.

Ihr Auftrag ist es, die Zentrale des Boghart zu finden, dort eine Bombe zu zünden und, soviel sei verraten, den Raum 1 lebend nach Süden zu verlassen.

Alle Eingaben finden in der untersten Zeile statt und werden mit <RETURN> bestätigt. Die Kommentartexte erscheinen zwei Zeilen höher. Räume werden immer in Blickrichtung dargestellt. Um Ihnen die Orientierung zu erleichtern, werden Richtung und Raumnummer angezeigt (Bild 1).

Der Parser (Analyse der Eingaben) erlaubt die Verwendung von einem oder zwei Wörtern.

Einwortbefehle

HILFE	ruft ein Hilfsmenü auf. Hier werden Befehle und Gegenstände aufgelistet.
QUIT	beendet das Spiel bei aussichtslosen Situationen.

Zur Bewegung dienen die Richtungsbefehle:

H	(Hoch)	N	(Nord)
R	(Runter)	O	(Ost)
W	(West)	S	(Süd)

Befehle bei zwei Eingaben

Das erste Wort stellt immer einen Befehl dar, das zweite Wort ist der betreffende Gegenstand.

VERSTECKE, VERLIER	mit diesen Befehlen, gefolgt von einem Gegenstand, werden Sachen abgelegt.
FINDE, NEHME, NIMM, SUCHE	im Raum befindliche Sachen werden in Besitz genommen (max. drei Gegenstände) und als Text links angezeigt (Bild 1).
ZERSTOERE	mitgetragene Gegenstände werden zerstört.
TOETE	einen Gegner. Mit Vorsicht zu genießen!

Kurzinfo: Future

Programmart: Grafik-Adventure
Spielziel: Suchen und sprengen Sie die Zentrale des herrschsüchtigen »Boghart«.
Laden: LOAD"FUTURE",8
Starten: nach dem Laden RUN eingeben
Steuerung: Tastatur
Benötigte Blocks: 62 Blocks Hauptprogramm und 1 Block Ladeprogramm
Programmautor: Herbert Großer

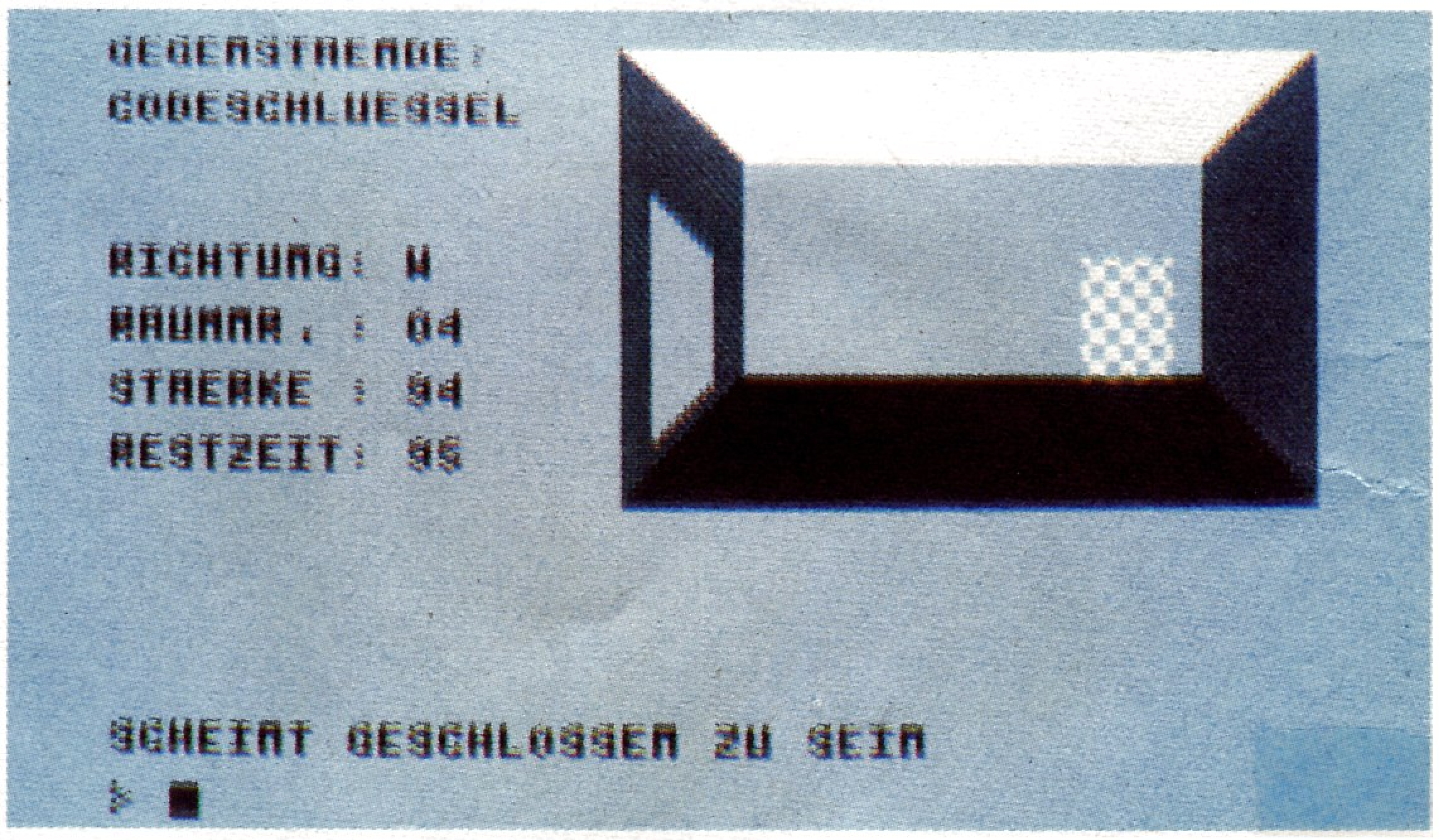
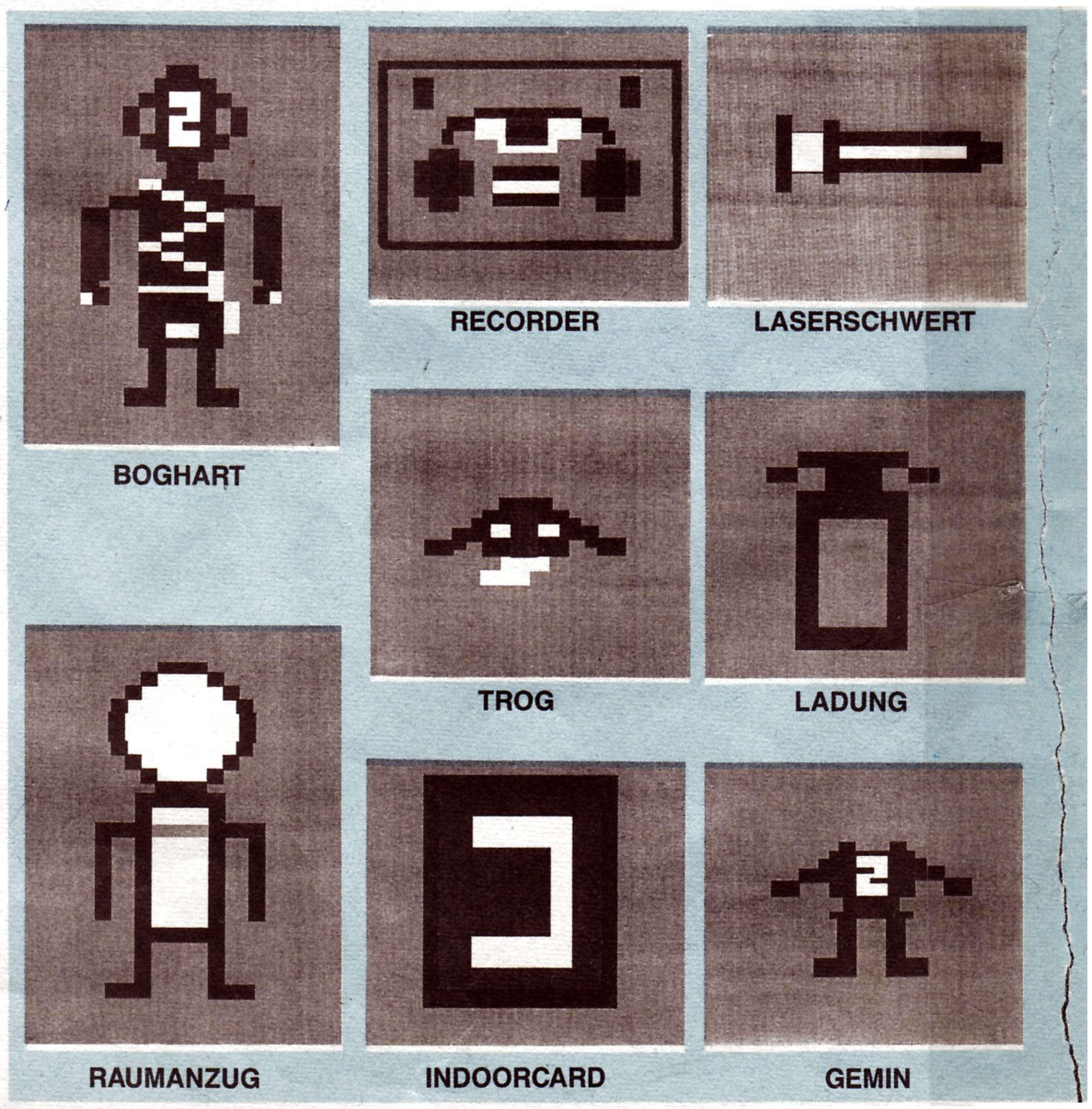


Bild 1. Raum 1 mit allen Anzeigen



LE

OEFFNE	einen nichtbeweglichen Gegenstand (z.B. Computer oder Tür).
BENUTZE	eine mitgetragene Sache.
ZUENDE	Bombe. Funktioniert nur, wenn die Voraussetzungen erfüllt sind.

Nichtbewegliche Gegenstände: COMPUTER, TUER

Bewegliche Gegenstände

INDOORCARD, OUTDOORCARD, BOMBE, ZUENDER, LASERSCHWERT, RECORDER, RAUMANZUG, LADUNG, CODESCHLUESSEL, SAUERSTOFFTANK.

Gegner

BOGHART	ist unangreifbar, kostet 35 Stärke-Punkte bei Kontakt.
GEMIN	wird nur mit dem »Laserschwert« überwunden, kostet bei Berührung 15 Stärke-Punkte.
TROG	ist harmlos. Er kostet Sie allerdings 10 Stärke-Punkte.

Alle im Raum befindlichen Gegenstände werden grafisch dargestellt (Bild 2).

Hilfreich bei der Suche ist ein Computer, dem Sie den Lageplan des Hauptquartiers entlocken können. Die Zentrale ist in diesem Plan grau gekennzeichnet. Doch Vorsicht! Sie können den Computer nur einmal »OEFFNEN«.

Geschlossene Türen sind mit einem Muster versehen. Beim Versuch Sie zu durchschreiten kostet das jeweils einen Stärke- und einen Zeit-Punkt. Beim Zählerstand »00« von Restzeit oder der Stärke bleibt Ihnen noch eine einzige Eingabe, dann ist Ihr Leben zu Ende. Starten Sie zu Ihrer schweren Mission. (gr)

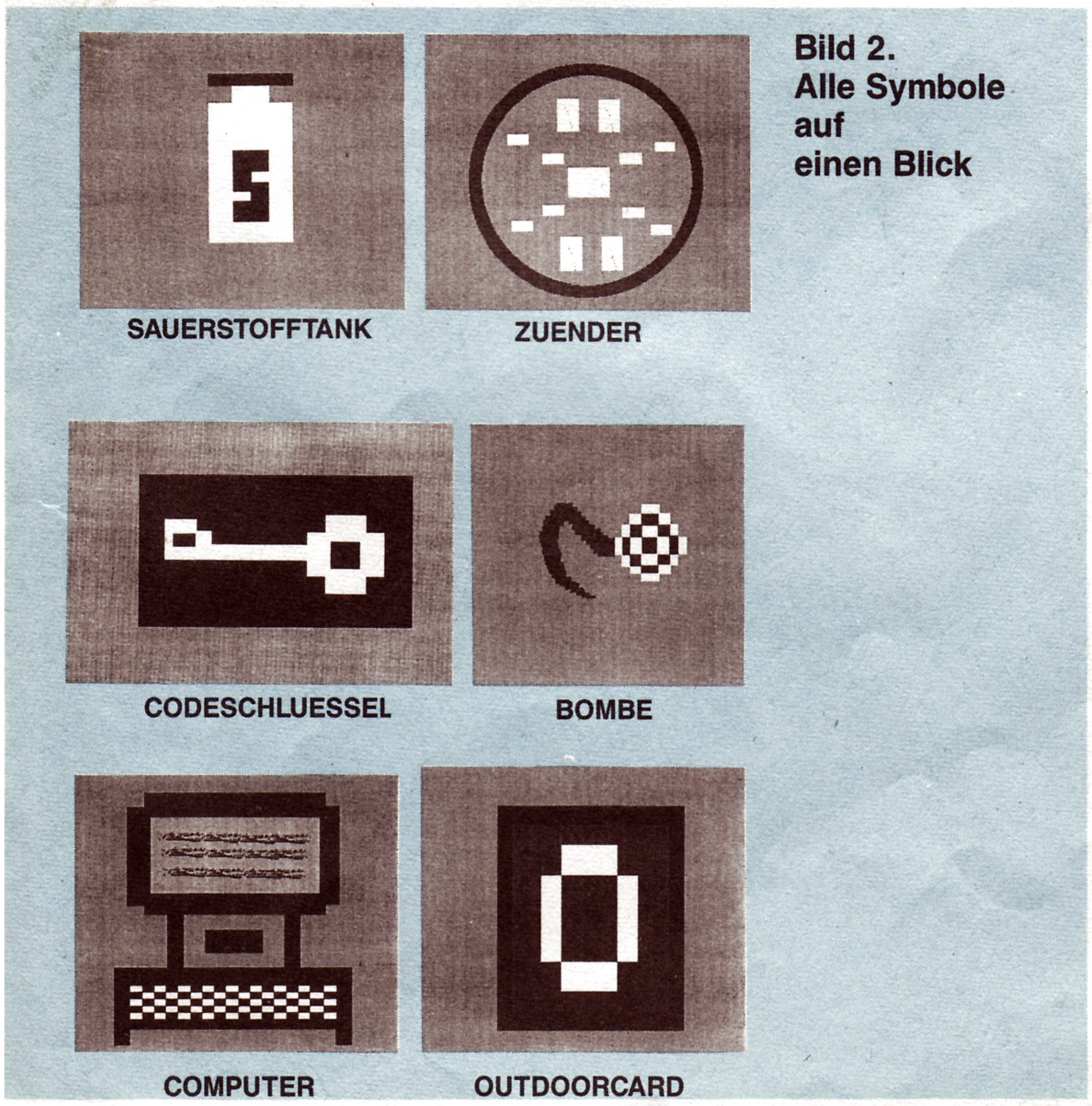


Bild 2. Alle Symbole auf einen Blick

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber
Redaktionsdirektor: Richard Kerler

Chefredakteur: Wolfram Höfler (hö)
Stellv. Chefredakteur: Gottfried Knechtel (kn) – verantwortlich für den redaktionellen Teil
Chef vom Dienst: Susanne Kirmaier
Redaktion: Andreas Greil (ag), Harald Beiler (bl), Herbert Großer (gr)
Mitarbeiter dieser Ausgabe: Nikolaus Heusler
Redaktionsassistenz: Brigitte Bobenstetter, Sylvia Derenthal, Helga Weber (202)
Telefax: 089/4613–778. **Hotline** (640): Montag bis Donnerstag 16 bis 17 Uhr, Freitag 11 bis 12 Uhr
Alle Artikel sind mit dem Kurzzeichen des Redakteurs und/oder mit dem Namen des Autors/Mitarbeiters gekennzeichnet

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten worden sein, muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt&Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt&Technik Verlag AG verlegten Publikationen. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Art-director: Friedemann Porscha
Titelgestaltung: Wolfgang Berns/Norbert Raab
Layout: Marian Schwarz, Andrea Miller
Bildredaktion: Janos Feitser (Ltg.), Sabine Tennstaedt; Roland Müller (Fotografie); Ewald Standke, Norbert Raab (Spritzgrafik); Werner Nienstedt (Computergrafik)

Anzeigenleitung: Philip Schiede (399) – verantwortlich für die Anzeigen
Telefax: 089/4613–775

Anzeigenverwaltung und Disposition: Monika Burseg (147)

Auslandsniederlassungen:
Schweiz: Markt&Technik Vertriebs AG, Kollerstr. 37, CH-6300 Zug, Tel. 042–440550/660, Telefax 042–415770, Telex: 862329 mut ch
USA: M&T Publishing Inc.; 501 Galveston Drive Redwood City, CA 94063, Telefon: (415) 366–3600, Telex 752–351
Österreich: Markt&Technik Ges. mbH, Große Neugasse 28, A 1040-Wien, Telefon: 0222/5871393, Telex: 047–132532

Anzeigen-Auslandsvertretung:
England: F. A. Smyth&Associates Limited, 23a, Aylmer Parade, London, N2 OPQ. Telefon:0044/1/3405058, Telefax: 0044/1/3419602
Israel: Baruch Schaefer, Haeskel-Str. 12, 58348 Holon, Israel, Tel. 00972–3–5562256
Taiwan: Aim International Inc., 4F.-1, No. 200, Sec. 2, Hsin-I Rd.; Taipei, Taiwan, R.O.C., Tel. 00886–2–7548631, –7548633, Fax 00886–2–7548710
Korea: Young Media Inc., C.P.O. Box: 6113, Seoul/Korea, Tel. 0082–2–7564819, /-7742759, Fax 0082–7575789

Vertriebsdirektor: Uwe Hagen

Vertriebsleitung: Helmut Grünfeldt (189)

Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: ip Internationale Presse, Hauptstätter Straße 96, 7000 Stuttgart 1, Tel. 0711/6483–110

Bezugsmöglichkeiten: Leser-Service: Telefon (089) 4613-366. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen.

Preis: Das Einzelheft kostet DM 16,–

Produktion: Technik: Klaus Buck (Ltg./180), Wolfgang Meyer (Stellv./887); Herstellung: Otto Albrecht (Ltg./917)

Druck: SOV Graphische Betriebe, Laubanger 23, 8600 Bamberg

Sonderdruck-Dienst: Alle in dieser Ausgabe erschienenen Beiträge sind in Form von Sonderdrucken zu erhalten. Anfragen an Reinhard Jarczok, Tel. 089/4613-185, Fax 4613-776.

Vorstand: Otmar Weber (Vors.), Bernd Balzer, Richard Kerler

Verlagsdirektor: Wolfram Höfler

Direktor Zeitschriften: Michael M. Pauly

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen: Markt&Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon 089/4613–0, Telex 522052, Telefax 089/4613–100

ISSN 0931–8933

Telefon-Durchwahl im Verlag:
Wählen Sie direkt: Per Durchwahl erreichen Sie alle Abteilungen direkt. Sie wählen 089/4613 und dann die Nummer, die in den Klammern hinter dem jeweiligen Namen angegeben ist.



ODY
Kampf der



SSEE

Bruderschaft

Als Auserwählter einer Bruderschaft erhalten Sie den Auftrag, das Land vom Zauberer Saruman zu befreien. Eine abenteuerliche Reise durch ein Fantasy-Land beginnt.

Vor 20 Jahren hat der mächtige Zauberer Saruman die Herrschaft über das Land Sosaria an sich gerissen. Die hiesige Bruderschaft ist nicht länger gewillt, seine Tyrannei zu erdulden. Sie erhalten den Auftrag, den Zauberer zu suchen und zu vernichten. Auf Ihrer Reise begegnen Sie Feinden, mit denen Sie schauerliche Kämpfe ausfechten werden. Aber auch Ordensbrüder stehen Ihnen mit Hilfestellungen zur Seite. Doch Vorsicht! Einige von ihnen sind Verräter, die Ihnen etliche Schwierigkeiten bereiten werden.

Ausgerüstet sind Sie mit 400 Einheiten Kraft, 350 goldenen Münzen und 50 Vorratseinheiten. Sie werden bald bemerken, daß diese Ausrüstung reichlich knapp bemessen ist. Zum Glück gibt es Dörfer und Städte, in denen Sie Kleider, Waffen und Vorräte kaufen können. Gold können Sie nach »alter Räuber-Sitte« in zahlreichen Kämpfen erobern.

Sie laden das Spiel von der beiliegenden Diskette mit:

```
LOAD"ODYSSEE",8
```

und starten es mit RUN.

Zuerst erstellen Sie einen Spieler. Hier sind 100 Punkte auf Stärke (Strength), Widerstandskraft (Stamina), Intelligenz (Intelligence) und Wissen (Wisdom) zu verteilen. Die Eingabe erfolgt zweistellig, beispielsweise »09« oder »10«.

Lassen Sie sich nicht irritieren, wenn kein Cursor sichtbar ist, zu Beginn befinden Sie sich in der ersten Zeile (Strenght). Bestätigen Sie die einzelnen Eingaben mit <RETURN>. In der untersten Zeile werden daraufhin die Restpunkte ange-

Tastendruck	Erklärung	Tastendruck	Erklärung
<@>	Bewegung nach Norden		anschließend Richtung eingeben
</>	Bewegung nach Süden	<X>	Das Schiff wird verlassen
<;>	Bewegung nach Westen	<U>	(UNLOCK DOOR) eine Tür aufschließen
<:>	Bewegung nach Osten	<M>	(MAGIC MISSILE) Starten der »Magic missile«
<A>	(ATTACK) Angriff eines Feindes; anschließend Richtung eingeben	<Z>	Anzeige des Inventars/durch beliebigen Tastendruck wird das Spiel fortgesetzt.
<T>	(TRANSACT) Verhandeln mit einem Ordensbruder; anschließend Richtung eingeben	<G>	(GOLD) zeigt an wieviel Goldmünzen Sie besitzen
<E>	(ENTER) betreten von Dörfern oder Städten; anschließend Richtung eingeben	<H>	(POWER) Anzeige der verbliebenen Kraft
<B>	(BOARD SHIP) Schiff betreten;	<R>	(RESTART GAME) Neues Spiel beginnen

Tabelle 1. Die Tastatureingaben und ihre Bedeutung auf einen Blick

zeigt. Sollten Sie zuviel Punkte (über 100) eingegeben haben, werden alle Kriterien auf Null gesetzt, und Sie beginnen wieder in der ersten Zeile (Strenght).

Als nächstes geben Sie einen fantasievollen Namen ein und bestätigen ihn mit <RETURN>. Danach dauert es etwa eine Minute bis das Spiel weitergeht. Nach dieser Wartezeit erscheint das Titelbild und der 2. Teil des Programms wird geladen. Danach befinden Sie sich mitten im Spiel.

Natürlich müssen Sie den Spieler auch bewegen. Dies geschieht normalerweise durch einen einzigen Tastendruck. Ausnahmen sind Befehle, die zusätzlich eine Richtungsanweisung benötigen (Tabelle 1). Die Bedeutung der Symbole zeigt Ihnen Tabelle 2. Im Laufe des Abenteuerspiels erreichen Sie **Städte (Town)** oder **Dörfer (Village)**:

Um Ihre Vorräte zu ergänzen, betreten Sie diese mit <E> (anschließend Richtungseingabe).

Danach sehen Sie eine Liste der im Ort befindlichen Geschäfte. Ebenso ist alles aufgeführt, was Sie in dieser Ortschaft noch erledigen können bzw. wie Sie den Ort wieder verlassen. Diese Liste könnte folgendermaßen aussehen:

1. Foodshop	(deutsch: Lebensmittelgeschäft)
2. Weaponshop	(deutsch: Waffengeschäft)
3. Armourshop	(deutsch: Bekleidungsladen)
4. Talk to the people	(deutsch: Gespräche mit den Einwohnern führen)
5. Exit the Town	(deutsch: die Stadt verlassen)

Sie wählen die einzelnen Punkte durch Drücken der entsprechenden Zifferntaste. In unserem Beispiel <1> für »Foodshop«, <2> um ein Waffengeschäft zu betreten (Weaponshop). Insgesamt wird angeboten:

Erwerb von Nahrung (Foodshop):

Nach dem Betreten des »Foodshop« erfahren Sie den Preis für 100 FOOD-Einheiten. Der Preis schwankt zwischen 32 und 52 Goldstücken. Ist der Preis annehmbar, so beantworten Sie die Frage nach Ihrer Kaufabsicht mit <Y>. Sie haben 100 Einheiten gekauft und erhalten die Anzeige der mitgeführten Lebensmittel. Sollten Sie nicht genügend Geld besitzen, erscheint eine entsprechende Fehlermeldung.

Kauf einer Waffe (Weaponshop):

Haben Sie den »Weaponshop« betreten, wird der Preis genannt und Sie entscheiden über den Kauf.

Kauf von Kleidung oder Rüstung (Armourshop):

Hier erscheint zunächst eine Liste der vorhandenen Bekleidungsstücke. Wählen Sie durch Drücken der entsprechenden Kennziffer einen Gegenstand aus. Sie erhalten danach einen Preisvorschlag und können diesen annehmen oder ablehnen. Besitzen Sie zu wenig Geld, erfolgt eine entsprechende Anzeige.

Gespräch mit den Einwohnern (Talk to the people)

Durch Anwahl dieses Menüpunktes erzählen Ihnen die Einwohner die Geschehnisse der Ortschaft.

Freunde und Feinde

Zu Ihrem Unglück gibt es viele feindlich gesonnene Krieger, die im Dienste des Zauberers stehen. Werden Sie in einen Kampf mit den Feinden verstrickt, so erscheint im Textfenster HIT oder MISS. HIT bedeutet einen Treffer gegen Sie. MISS traf daneben. Wie oft Sie Schläge einstecken müssen, hängt von der Dauer des Kampfes, Ihrer Widerstandsfähigkeit (STAMINA) und Ihrer Kleidung ab.

Falls Sie keine Nahrung (FOOD) oder keine Kraft (POWER) mehr haben, ist das Spiel verloren. »FOOD« verbrauchen Sie durch Ihre Bewegungen (pro Schritt 0,2 Einheiten). Durch Treffer in den Kämpfen verlieren Sie Kraft.

Hin und wieder begegnen Sie gut gesonnenen Ordensbrüdern. Einige teilen Ihnen Geheimwörter mit, die andere von Ihnen wissen wollen. Andere beurteilen Sie nach Ihrer Intelligenz (INTELLIGENCE) oder nach Ihrem Wissen (WISDOM). Doch Vorsicht, es gibt Ordensbrüder, die es auf Ihr Gold abgesehen haben.

Hilfsmittel

Um Ihre Aufgabe zu lösen, benötigen Sie Hilfsmittel. Diese erhalten Sie in Kämpfen, Dörfern oder Städten und von den Ordensbrüdern. Das Drücken der Taste <Z> listet sie auf. Folgende Hilfsmittel stehen Ihnen zur Verfügung:

BLUE TASSLE: Matrosenanzug

Ohne Matrosenanzug läßt Sie die Besatzung eines Schiffes nicht an Bord.

MAGIC MISSILE: magische Waffe

Bringt Sie in die Lage, alle Feinde zu vernichten. Leider werden auch alle sichtbaren Freunde mit vernichtet. Pro Person erhalten Sie fünf Goldstücke.

KEY: Schlüssel

Öffnet Ihnen die entsprechenden Türen.

Kurzinfo: Odyssee

Programmart: Grafisches Abenteuerspiel
Spielziel: Besiegen Sie den Zauberer
Laden: LOAD"ODYSSEE",8
Starten: Nach dem Laden RUN eingeben
Steuerung: Tastatur
Benötigte Blocks: 233 Blocks
Programmautor: Jan Geiszelmann

Erläuterung der Grafik

Sie bewegen sich normalerweise nur auf Gras. Außer den Steinmauern können alle Hindernisse überwunden werden. Die Energiemauern nehmen Ihnen 100 »Power-Einheiten« ab.

Bedeutung	Normales Zeichen
Spielfigur	!
Der Ordensbruder	"
Der Feind	#
Eine Stadt	$
Wasser	%
Ein Schiff	&
Ein Dorf	/
Gras	(
Fels	)
Lava	+
Energiemauer	*
Zauber Saruman	↑
Der dunkle Turm	←
Mauerstein	]
Schloß eines Lords	,
Tür	

Tabelle 2. Die einzelnen Symbole und ihre Bedeutungen

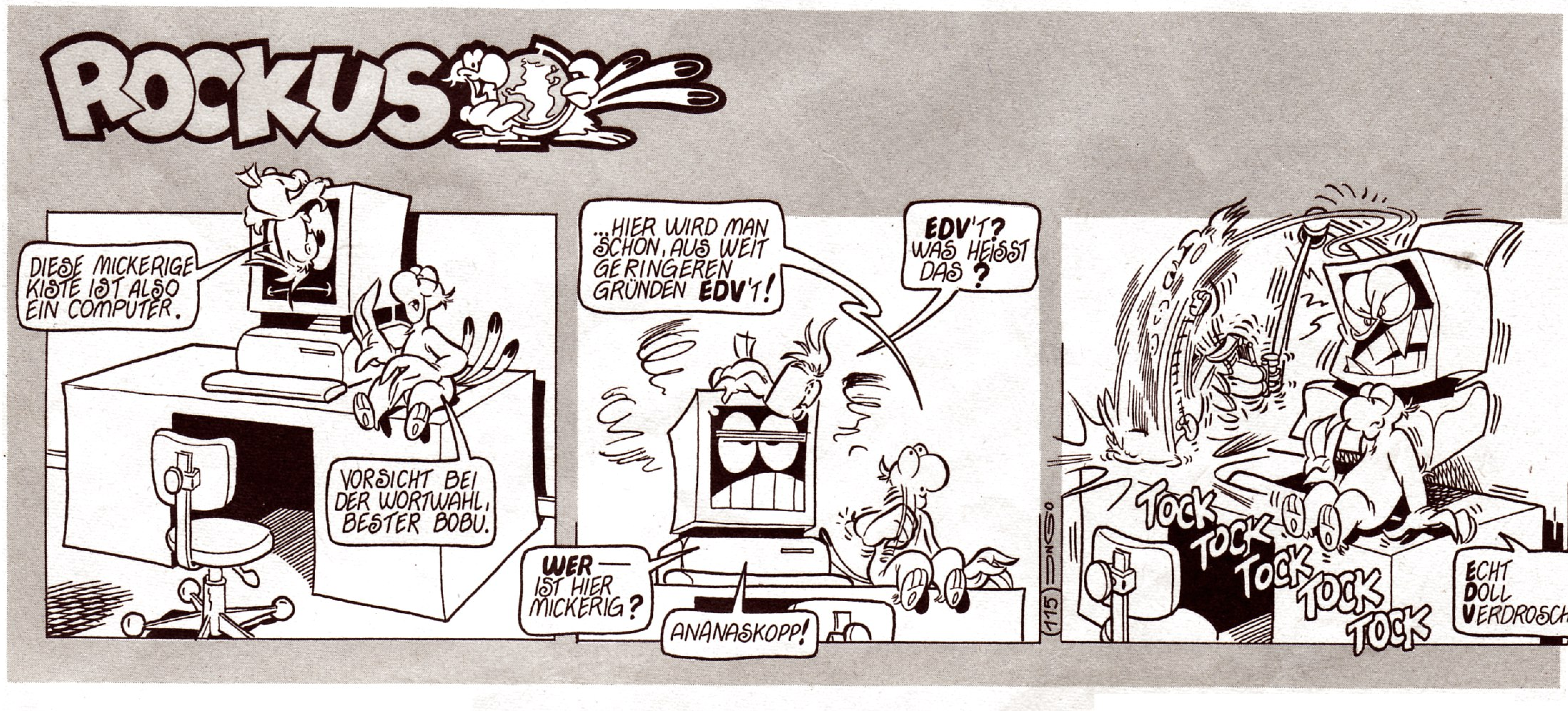
GOLDEN KEY: goldener Schlüssel
Hierzu soll an dieser Stelle nichts verraten werden.

SWIMWEST: Schwimmweste
Der Besitz einer Schwimmweste erlaubt es, zwölf Felder weit zu schwimmen. Danach verschwindet die Schwimmweste auf Nimmerwiedersehen.

BOOTS: Schuhe
Schuhe ermöglichen ihnen einen Lauf über zwei Felder auf felsigem Grund. Danach sind sie verschlissen.

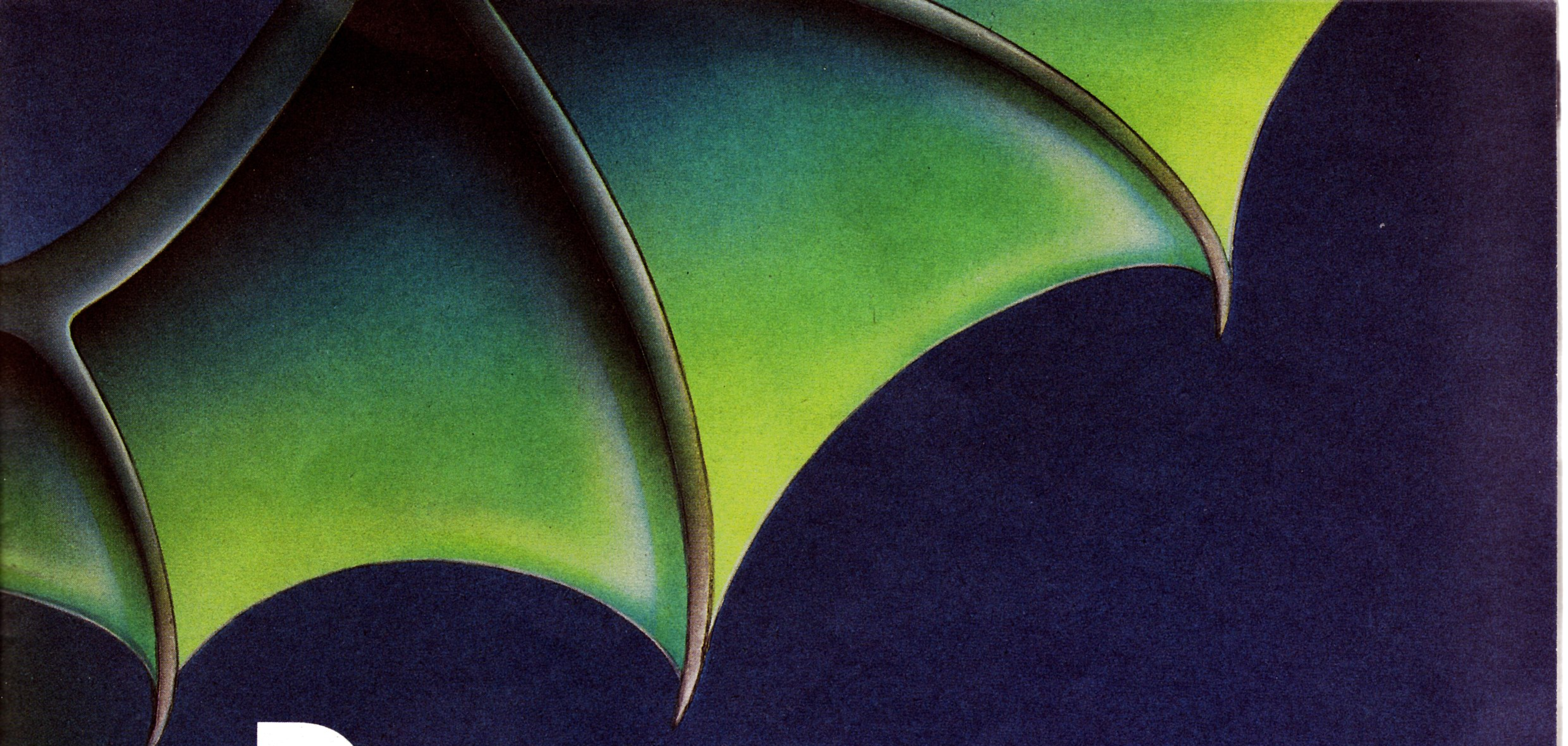
MARK OF FIRE:
Mit MARK OF FIRE können Sie durch LAVA gehen.

(Jan Geißelmann/gr)





Ein spannendes Abenteuerspiel
fesselt den C64-Freak
mehr als alles andere.
Sie haben eine tolle Idee dazu,
wissen aber nicht,
wie man diese umsetzt.
Nach diesem Kurs sind Sie in die Lage,
Super-Adventures selbst zu programmieren.



Das Abenteuer lockt!

Eine unheimliche Stille liegt in der Luft... Sie befinden sich in einem großen Raum, in dessen Mitte auf dem kalten Steinboden ein roter Teppich liegt. Eine Fackel, die an der Wand in einer Halterung steckt, beleuchtet spärlich den Raum. Am Boden liegt ein zwei Meter langes stabiles Holzbrett. Plötzlich fällt ein schweres Fallgitter hinter Ihnen herunter und versperrt den einzigen Ausgang.

Alle Versuche, das Fallgitter zu heben, scheitern. Mit großem Schrecken hören Sie ein lautes Rattern – die Zimmerdecke bewegt sich langsam, aber sicher nach unten. Erst jetzt bemerken Sie die spitzen Eisenstangen, die von der Decke herunterragen! Sie erinnern sich sofort an den Zauberring, der einem Monster gehörte, das Sie getötet haben. Sie greifen in Ihre Tasche, holen den Ring heraus und stecken ihn sich an den Finger. Sie drehen den Ring in der Hoffnung, daß er Sie durch seine Zauberkraft befreien kann – nichts passiert. Inzwischen hat sich die Decke bis auf 3 m Höhe gesenkt. Ihr nächster Gedanke ist das stabile Brett, Sie klemmen es zwischen Boden und Decke. Sie atmen auf, als Sie feststellen, daß das Brett die Decke, wenn auch nicht das Rattern zum Stillstand bringt. Die Freude findet ein jähes Ende, als das Brett dem ungeheuer großen Druck der Decke nachgibt und zerbricht. Die Decke ist jetzt so weit unten, daß Sie nicht mehr aufrecht stehen können. In letzter Sekunde kommt Ihnen der rettende Gedanke: Sie stürzen auf den kleinen roten Teppich zu...

Dies könnte eine von vielen Action-Szenen eines Abenteuerspiels sein, das Sie bald selbst programmieren werden.

Was ist eigentlich ein Adventure?

Eine Antwort darauf könnte z.B. lauten: »das Gegenteil eines Ballerspiels«. Während beim »Weltraum-« oder »Grafik«-Action-Spiel flinke Finger und ein starrer Blick zum Bildschirm das Wichtigste sind, braucht man für ein Adventure einen scharfen Verstand.

Wer ein Adventure spielen will, muß viel Fantasie und Einfühlungsvermögen mitbringen. Hier stellt der Computer ein Fenster in eine andere Welt dar, deren Grenzen und Gesetze vom Programmierer gesetzt werden. Das Prinzip eines Adventures läßt sich am besten anhand eines Rollenspiels erklären. Sie sind besonders in England unter dem Titel »Dungeons & Dragons« (Höhlen und Drachen) verbreitet.

Wie ist ein Rollenspiel aufgebaut?

Da gibt es einen »Dungeon-Master«. Das bedeutet so viel wie »Höhlenmeister«, »Spielmeister«. Er erstellt das Spiel und ist der Spielleiter. Den einzelnen Spielern werden Charaktere wie Zauberer, Fee oder Zwerg etc. zugeteilt. Außerdem gibt es Nichtspieler-Charaktere. Das sind z.B. Monster und Kreaturen, die in dem Spiel auftreten. Sie werden ebenfalls vom Höhlenmeister gelenkt. Der Spielleiter ist der absolute Schiedsrichter. Er verteilt während des Spiels die Punkte. Falls sich im Spiel eine besondere Situation ergibt, muß er improvisieren, um den Spielern gerecht zu werden.

Viel Fantasie und Ideenreichtum sind vom Spielmeister gefordert. Der Ablauf eines Rollenspiels:

Höhlenmeister: Ihr befindet Euch in einer großen, trockenen Höhle mit Fackeln an den Wänden. Was macht Ihr jetzt?

Die Spieler beraten sich.

Sprecher der Spieler: Wir untersuchen die Höhle genauer.

Der Höhlenmeister sieht in seinen Plänen und Notizen nach, ob in der Höhle irgend etwas zu finden ist.

Höhlenmeister: Ihr entdeckt eine Geheimtür. Was nun?

So finden Sie die Programme auf der Diskette

DISKETTE SEITE 1

```
0   "64'ER SONDERH.52" S1 2A
0     "----------------" DEL
0     "----------------" DEL
0     "-----SPIELE-----" DEL
0     "----------------" DEL
0     "----------------" DEL
123   "ODYSSEE"          PRG
28    "CREATER"          PRG
82    "WORLD"            PRG
0     "----------------" DEL
114   "MARIO"            PRG
3     "SCR/1"            PRG
0     "----------------" DEL
8     "R.A.M.S."         PRG
1     "RAMS LOAD"        PRG
122   "RAMS MAIN"        PRG
1     "RAMS SOUND"       PRG
9     "RAMS CHARS"       PRG
4     "RAMS ROUTINEN"    PRG
1     "RAMS 0"           SEQ
0     "----------------" DEL
50    "QUASIMODO"        PRG
0     "----------------" DEL
107   "ZEITTUNNEL"       PRG
0     "----------------" DEL
0     "----DISKETTE----" DEL
0     "---BEIDSEITIG---" DEL
0     "----BESPIELT----" DEL
0     "----------------" DEL
11 BLOCKS FREE.
```

DISKETTE SEITE 2

```
0   "64'ER SONDERH.52" S2 2A
0     "----------------" DEL
0     "----------------" DEL
0     "-----SPIELE-----" DEL
0     "----------------" DEL
0     "----------------" DEL
138   "ADVENTURE 2000"   PRG
1     "SCROLL.MC"        PRG
0     "----------------" DEL
1     "FUTURE"           PRG
62    "MSG.FUT"          PRG
0     "----------------" DEL
0     "----------------" DEL
0     "---GRUNDLAGEN---" DEL
0     "----------------" DEL
0     "----------------" DEL
1     "LISTING 1"        PRG
2     "LISTING 2"        PRG
3     "LISTING 3"        PRG
2     "LISTING 4"        PRG
3     "LISTING 5"        PRG
6     "LISTING 7"        PRG
2     "LISTING 8"        PRG
9     "LISTING 6"        PRG
3     "LISTING 9"        PRG
1     "LISTING 10"       PRG
12    "LISTING 11"       PRG
1     "LISTING 12"       PRG
1     "LISTING 13"       PRG
1     "LISTING 14"       PRG
1     "LISTING 15"       PRG
1     "LISTING 16"       PRG
14    "LISTING 17"       PRG
1     "LISTING 18"       PRG
1     "LISTING 19"       PRG
1     "LISTING 20"       PRG
1     "LISTING 21"       PRG
17    "LISTING 22"       PRG
1     "LISTING 23"       PRG
1     "LISTING 24"       PRG
1     "LISTING 25"       PRG
23    "LISTING 26"       PRG
5     "GRAFIK-DESIGNER"  PRG
0     "----------------" DEL
0     "----------------" DEL
0     "------ENDE------" DEL
0     "----------------" DEL
0     "----------------" DEL
338 BLOCKS FREE.
```

WICHTIGE HINWEISE

zur beiliegenden Diskette:

Aus den Erfahrungen der Sonderhefte 46 und 47 wollen wir ein paar sinnvolle Tips an Sie weitergeben:

1 Bevor Sie mit den Programmen auf der Diskette arbeiten, sollten Sie unbedingt eine Sicherheitskopie der Diskette anlegen. Verwenden Sie dazu ein beliebiges Backup-Programm, das eine komplette Diskettenseite kopiert.

2 Auf der Originaldiskette ist wegen der umfangreichen Programme nur wenig Speicherplatz frei. Dies führt bei den Anwendungen, die Daten auf die Diskette speichern, zu Speicherplatzproblemen. Kopieren Sie daher das Programm, mit dem Sie arbeiten wollen, mit einem File-Copy-Programm auf eine leere, formatierte Diskette und nutzen Sie diese als Arbeitsdiskette.

3 Die Rückseite der Originaldiskette ist schreibgeschützt. Wenn Sie auf dieser Seite speichern wollen, müssen Sie vorher mit einem Diskettenlocher eine Kerbe für die Rückseite der Diskette machen, um den Schreibschutz zu entfernen. Probleme lassen sich von vornherein vermeiden, wenn Sie die Hinweise unter Punkt 2 beachten.

ALLE PROGRAMME

aus diesem Heft



HIER

SONDERHEFT

53 VORSCHAU

Einen repräsentativen Querschnitt von zwölf herausragenden Software-Produkten haben wir für das Jubiläumsheft 53 »5 Jahre Sonderhefte« zusammengestellt:

Mit »Giga-Ass« und »Promon« erhalten Sie einen komfortablen Makroassembler und einen Maschinensprache-Monitor mit fantastischen Funktionen.

Eine Grafik-Befehlserweiterung des Basic 2.0 mit tollen Möglichkeiten ist »Grafik 2001«.

»Datec 3.1« liefert Ihnen eine Dateiverwaltung für höchste Ansprüche.

Der »Sound-Monitor« stellt einen Editor zum Komponieren eigener Musikstücke zur Verfügung, die Produkten professioneller Tonstudios nahezu ebenbürtig sind.

»Proterm V6.0« erschließt Ihnen die faszinierende Welt der Mailboxen und Telekommunikation.

Selbstverständlich kommen auch die Spiele-Freaks nicht zu kurz:
- »Crillion«, ein brandheißes Geschicklichkeits- und Strategiespiel.
- Die packende Eishockey-Simulation »Bully« sorgt für spannende Wettkämpfe.



Das Sonderheft 53 liegt ab 20.4.90 an Ihrem Kiosk.

Fortsetzung von Seite 17

Die Spieler beraten sich erneut.

Sprecher der Spieler: Wir öffnen die Tür und lassen unseren stärksten Mann vorausgehen.

An diesem Beispiel können Sie die Aufgaben des Höhlenmeisters erkennen. Ab sofort wollen wir den C64 als Höhlenmeister arbeiten lassen. Dazu geben wir ihm Pläne, Tabellen etc., aus denen er ersehen kann, welche Antworten er geben soll. Das Problem dabei ist, daß der Computer an den Plänen manipulieren und während des Spiels improvisieren soll, damit keine Langeweile beim Spieler aufkommt.

Bevor wir uns an dieses Problem wagen, müssen wir lernen, wie man Pläne bzw. ein Spielkonzept erstellt. Sie können den Computer getrost ausschalten, sich in einen bequemen Sessel zurücklehnen und aufmerksam die folgenden Kapitel lesen...

Spielplatz = Speicherplatz

Im Prinzip gibt es zwei Faktoren, die uns beim Erstellen von Abenteuerspielen einschränken: Zum einen die Grenzen unserer Fantasie und zum anderen die Speicherkapazität des C64.

Wieviel Abenteuer paßt in 38911 Byte?

Oberstes Gebot: Schränken Sie Ihre Fantasie niemals aus Angst vor Speichermangel ein. Wenn Sie die Idee zu einem gigantischen Spiel haben, dann kürzen Sie es niemals so lange, bis es in den C64 paßt. Teilen Sie es in mehrere Programme auf, die nacheinander geladen und gespielt werden. Die Antwort auf die Frage nach dem Speicher hängt hauptsächlich von der Art des Spiels ab. Wir unterscheiden:

Spiele mit überwiegend Grafik (Grafik-Adventure)
Spiele ausschließlich mit Text (Text-Adventure)

Bei Grafik-Abenteuerspielen findet man für jeden Raum ein Bild. Unter diesem Bild ist ein Textfenster zur Befehlseingabe, ein Beispiel hierfür ist das Adventure »Zauberschloß« aus dem 64'er-Sonderheft 42. Bei Textabenteuerspielen findet man anstelle von Bildern lange Texte, die jeden Raum ausführlich beschreiben. Das Bild entsteht in der Fantasie des Spielers, der den Text liest.

In diesem Fall werden neben Kenntnissen der Programmiersprache auch schriftstellerische Fähigkeiten vom Entwickler eines Adventures verlangt — mit ein Grund, warum Abenteuerspiele mit Grafik entstanden sind.

Als ich mein erstes Adventure schrieb, war ich besonders darauf bedacht, viele Räume zu haben. Ich war der Meinung, daß dies für ein sehr gutes Adventure unbedingt notwendig sei. Ich bemerkte bald, daß ich hier völlig falsch lag - das Resultat war ein Spiel mit 200 (!) Räumen, aber kaum Action. Alles, was man konnte, war herumlaufen und laufen und laufen.

Wir wollen unsere anfänglich gestellte Frage deshalb erweitern. Angenommen, wir müssen uns exakt mit dem maximalen Speicherplatz des C64 begnügen: Wie sieht in so einem Fall ein gutes Adventure aus, und wie bringt man es auf 38 KByte? Das Wesentliche ist der Komfort der Befehlseingabe und deren Analyse. Untrennbar damit hängt der Wortschatz des Spiels zusammen. Ein Beispiel:

Stellen Sie sich ein Abenteuerspiel mit dem Titel X vor.
Wenn X einen Befehl erwartet, fragt es »WAS NUN?«.
Wir geben folgendes ein: »Nimm Schwert.«
X antwortet uns: »Sie nehmen das Schwert.«

Heißt die Befehlseingabe aber »Nimm das Schwert«, dann antwortet X: »Ich kenne (das) nicht.« oder: »Sie können (das) nicht nehmen«, »Das geht nicht« usw.

Was haben wir falsch gemacht?

Für uns menschliche Wesen ist »Nimm Schwert« und »Nimm das Schwert« gleichermaßen verständlich, die zweite Formulierung zudem besseres Deutsch. Warum versteht uns der Computer nicht?

Ganz einfach - die Befehlsanalyse von X sieht so aus:

X versteht nur Befehle, die aus zwei Wörtern (Verb und Objekt) bestehen. Geben wir die Anweisung »Nimm das Schwert« ein, interpretiert X das Wort »das« als Objekt der Befehlsfolge. X sieht in seiner Objekttabelle nach und kann »das« nicht finden. X versteht den Befehl nicht.

Spiel ohne Grenzen

An diesem simplen Beispiel erkennen Sie, was mit guter Befehlsanalyse gemeint ist: Das Spiel soll längere Sätze wie »Nimm das Schwert, den Ring und das Brett und gehe nach Norden« verstehen. Davon später mehr.

Ein weiteres Kriterium für ein gutes Spiel ist: Der Spieler sollte bedeutend mehr Möglichkeiten haben, als nur Gegenstände zu nehmen, zu verlieren und herumzulaufen.

Wenn wir ein gutes Abenteuerspiel schreiben wollen, so müssen wir dem Spieler die Möglichkeit bieten, sich nach Lust und Laune in unserer Fantasiewelt austoben zu können. Vor allen Dingen dann, wenn der Spieler etwas macht, das mit der Lösung des Spiels nichts zu tun hat.

Stellen wir uns einen Raum mit einer großen Kiste und einem Fenster vor.

Wir sollten dem Spieler folgende Dinge ermöglichen:

- Kiste öffnen und schließen (wenn wir wollen, muß er dafür einen Schlüssel haben).
- Gegenstände in die Kiste legen.
- Kiste mitnehmen, falls sie nicht zu schwer ist.
- sich in der Kiste verstecken (wenn sie groß genug ist).
- Kiste verschieben, falls sie zu schwer zum Tragen ist.
  Achtung: Der Spieler kann nur in die Kiste gehen, wenn alle Gegenstände, die er besitzt, mit in die Kiste passen. Ansonsten muß er sie vorher ablegen.
- Fenster öffnen und schließen oder hinaussehen.
- aus dem Fenster springen.

Sie sehen, wie viele Möglichkeiten in der Ausgangssituation (Kiste und Fenster) stecken.

Lebendige Personen sind in einem guten Adventure nie fehl am Platz. Allerdings ist es langweilig, nur auf Taubstumme zu treffen. Kommen Personen im Spiel vor, sollten sie mehr können, als nur herumstehen oder wortlos hinter dem Spieler herrennen. Ein gutes Beispiel für »selbstdenkende« Spielcharaktere finden wir im englischen Adventure »The Hobbit« von Melbourne House. Wer es kennt (wahrscheinlich jeder Adventure-Freak), weiß, was Thorin, Gandalf, Elrond, Bard, der Butler oder Smaug, der Drache, im Spiel alles selbständig treiben. Dieses gute Adventure bietet aber auch Grund zur Kritik:

Sagt man seinem Freund Elrond »Lies die Karte«, so kann es vorkommen, daß dieser einfach »nein« sagt. Wird man von Monstern gefangen und wartet auf Rettung durch einen Freund, kann es passieren, daß man heute noch wartet...

Fassen wir zusammen, was ein gutes Adventure auszeichnet:

1. Der Spieler sollte möglichst viele (auch unsinnige) Möglichkeiten haben, damit ihm die Grenzen des Spiels nicht auffallen.
2. Das Spiel muß einen großen Wortschatz und eine großzügige Befehlsanalyse besitzen, d.h. es soll Worte nicht nur erkennen, sondern den Sinn eines ganzen Satzes verstehen.
3. Das Abenteuerspiel sollte zu jedem Raum einen ausführlich beschriebenen Text haben. Grafiken können diesen Text ab und zu unterstützen - doch darf nicht sein, daß ein Spieler erst rätseln muß, was die Grafik darstellt. Grafiken sind nur Ergänzungen zu einem guten Text - sie können ihn nicht völlig ersetzen.
4. Wenn im Adventure Personen (Charaktere) auftreten, soll-

ten sie einen eigenen Willen haben, sprechen und verstehen können.

Nehmen wir als Beispiel erneut den Raum mit Kiste und Fenster:

Ein selbstdenkender Spielcharakter, z.B. ein Geist, kommt in das Zimmer. Er hat im dem Fall dieselben oder ähnliche Möglichkeiten wie der Spieler.

Der Geist sieht die Kiste und beschließt, sich darin zu verstecken. Er öffnet den Kasten, steigt hinein und schließt ihn von innen. Wenn jetzt der Spieler den Raum betritt, bieten sich viele neue Möglichkeiten.

Aktion 1. Der Spieler öffnet die Kiste:
- Der Geist erschreckt und flieht.
- Der Geist erschreckt den Spieler zu Tode.
- Die Kiste ist leer. Warum? Bei einem Geist ist alles möglich.

Aktion 2. Der Spieler öffnet die Kiste nicht, bleibt aber im Zimmer:
- Der Geist kommt aus der Kiste heraus...
- Der Spieler hört ein Schluchzen und Weinen. Warum? Der Geist hat sich selbst in der Kiste eingeschlossen und bittet Sie, ihn herauszulassen.

Aktion 3. Der Spieler hilft dem Geist:
- Der Geist belohnt den Spieler großzügig.
- Der böse Geist tötet den Spieler trotzdem.

Aktion 4. Der Spieler öffnet die Kiste und erwischt den Geist beim Flirten mit einer Hexe.
- Der Geist errötet und löst sich in Luft auf.
- Die Hexe belegt den Spieler mit einem Zauberbann, den er mit Hilfe eines Losungswortes aufheben kann.
- Der Geist schlägt die Kiste wieder zu.

Sie sehen, wie viele Möglichkeiten in der Situation »Kiste, Fenster, Geist und Spieler« stecken. Sicher haben Sie bereits eigene Ideen ausgeheckt.

Weitere Anregungen gefällig? Stellen Sie sich eine zusätzliche Person in unserer Ausgangssituation vor. Was passiert,
- wenn der Spieler auf diese Person trifft?
- wenn diese Person dem Gespenst begegnet?
- wenn sich die neue Spielfigur (z.B. eine Hexe) und der Geist verbünden, um gemeinsam gegen den Spieler vorzugehen?

Unsere Frage, wieviel Adventure in 38 KByte paßt, ist noch nicht beantwortet. Die bisherigen Ausführungen sollten Ihnen begreiflich machen, warum eine konkrete Antwort unmöglich ist. Gerade als Einsteiger in die Adventure-Programmierung ist man kaum in der Lage, abzuschätzen, wieviel Speicher für das geplante Spiel benötigt wird. Dieses Abschätzen will gelernt sein. Doch dazu mehr im Abschnitt über die Programmentwicklung.

Ohne Drehbuch kein Film

Das Schreiben von Abenteuerspielen kann man am besten mit dem Drehen eines Films vergleichen. Während am Entstehen eines Kino-Films unzählige Leute mitarbeiten, müssen Sie alle Arbeiten ganz allein verrichten. Sie sind Drehbuchautor, Regisseur und Kameramann in einer Person.

Lange bevor ein Filmregisseur zu drehen beginnt, braucht er die Story, worum es in dem Film gehen soll. Auch der Programmierer eines Adventures muß eine Idee aufgreifen, aus der später das Spiel entsteht. Das ist vermutlich der schwierigste Teil. Es ist nicht leicht, ein absolut neues und möglichst originelles Spielthema zu finden. Denken wir dabei an die zahlreichen Spiele, bei denen eine Burg oder ein Schloß im Mittelpunkt steht: Zauberschloß, Burg des Schreckens, Schloß des Grauens usw. Viele Spiele finden in Höhlen (engl. »Caves«) statt. Darum sollten Sie bei ihren Überlegungen ein Spiel anstreben, das sich im gesamten Abenteuerland und nicht nur in einer Burg abspielt.

Im Prinzip sind folgende Spielarten möglich:

Märchen und Legenden
Science-fiction
modernes Adventure

Unter »Märchen« versteht man ein Spiel mit Zauberern, Hexen, Drachen usw. Bei »Science-fiction« könnte es sich um ein Weltraum- oder ein Zukunfts-Abenteuer handeln. Ein »modernes Adventure« spielt in unserer Zeit. Selbstverständlich können sich diese Themen überschneiden (z.B. die Reise mit einer Zeitmaschine). Ich möchte Ihnen als Beispiel einige Gedanken für eine Spiel-Story präsentieren.

Von der Idee zum Spiel

Nehmen wir an, ich möchte ein Spiel schreiben, das in der Vergangenheit spielt. Ich brauche zu Beginn des Adventures eine Zeitmaschine, mit der der Spieler in die Vergangenheit reisen kann. Da es heutzutage noch kein Gerät für Reisen in die Zeit gibt, muß ich den Anfang des Spiels in die Zukunft verlegen. Unser erster Spielgedanke könnte sein:

Im Jahre 2000 wird eine Zeitmaschine mit zwei Mann Besatzung in die Vergangenheit geschickt. Der Spieler gehört zur Besatzung der Zeitmaschine. Damit er nicht alleine ist, geben wir ihm einen Begleiter, einen Wissenschaftler, mit auf den Weg (der Professor wird im Programm als selbstdenkender Spielcharakter behandelt). Selbstverständlich müssen wir dem Spieler eine Aufgabe stellen. Er könnte herausfinden, warum die Dinosaurier seinerzeit ausgestorben sind.

Damit ist unser erster Gedanke - die Ausgangssituation - abgeschlossen. Unser Spieler fliegt in die Vergangenheit. Jetzt müssen Sie sich ausdenken, was ihn dort erwartet. Er könnte auf Steinzeitmenschen oder Außerirdische treffen. Neandertaler würden ihn gefangennehmen. Sie müssen sich voll auf Ihre Fantasie stützen.

Lassen Sie Ihrer Vorstellungskraft freien Lauf. Wenn die Gedankenblitze greifbare Formen angenommen haben, sollten Sie prüfen, ob völlig unrealistische Situationen darunter sind. Verzichten Sie lieber auf solche Ideen.

Bevor das Spiel in ein Programm umgesetzt wird, muß es in Form von Skizzen und Tabellen (das »Drehbuch«) nahezu fertig vorliegen. Viele Programmierer setzen sich an den Computer und schreiben drauflos, nach dem Motto: erst Tippen, dann Denken. Auf diese Weise entstehen Spiele, die so viele stilistische und programmbedingte Fehler enthalten, daß sich das Spielen nicht lohnt. Sie wandern in die Schublade und werden schnell vergessen. Außerdem geht beim Programmieren nach einiger Zeit der Überblick verloren, so daß Korrekturen mühsam werden. Der Programmierer verliert die Lust, weiterzumachen.

Einige Leser könnten einräumen, daß sie ihre Programme gut strukturieren und ein Überblick jederzeit gewährleistet ist. Das mag stimmen, soweit es sich um die Programmierung eines Schießspiels handelt. Aus eigener Erfahrung kann ich Ihnen versichern: Es ist unmöglich, ein 38-KByte-Adventure bis auf die letzte Variable zu strukturieren - irgendwann findet man sich selbst im besten Programmaufbau nicht mehr zurecht. Außerdem sind strukturierte Programme viel länger als herkömmliche und verbrauchen eine Menge Speicherplatz.

Niemand kann von Ihnen verlangen, jedes Adventure vor der Umsetzung zum Programm bis ins allerletzte Detail auszuarbeiten. Sie sollen lernen, welche Unterlagen man stets beim Programmieren neben sich liegen haben muß, um problemlos und schnell Änderungen am Spiel vornehmen zu können. Generell empfehle ich, ein Heft (DIN-A4-Format) zu führen, in dem alle wichtigen Informationen und Tabellen zum Spiel enthalten sind.

Bücher als Ideen-Lieferanten

Am Anfang unserer Arbeit steht die Idee. Gut gesagt, aber woher nehmen? Hierzu bietet sich an, den Software-Markt zu durchforsten und einige professionelle Adventures zu spielen. Gute Abenteuerspiele, die wir Ihnen empfehlen können, sind: »Zak McKracken« von Rushware oder »Logan« aus »64'er-Disc«, Ausgabe 1/90. Es ist klar, daß Sie die darin enthaltenen Ideen nicht in Ihren eigenen Spielen kopieren sollen – es genügt, daraus zu lernen, wie andere Autoren gewisse Ideen verwirklicht haben. Es ist kein Beinbruch, wenn Sie an ein extrem schlechtes Adventure geraten. Sie werden aus den Fehlern lernen und sie in eigenen Abenteuerspielen vermeiden.

Unter »Abenteuer« sollte man »Erleben« verstehen. Der Sinn eines Adventures besteht nicht alleine darin, einen Zauberer zu besiegen oder Schätze zu finden. Es soll ein Spiel sein, das der Spieler selbst beeinflussen kann und dabei einiges erlebt. Hier noch einige Beispiele

Spielidee 1: Der Löwe ist los

Ein kleiner Wanderzirkus kommt in die Stadt. In der Nacht bricht ein gefährlicher Löwe aus und macht die Stadt unsicher.

Die Aufgabe des Spielers besteht darin, den Löwen zu finden und zu fangen.

Spielidee 2: Juwelenraub auf der »Queen Mary«

Auf einem riesigen Passagierdampfer reist eine Gräfin mit. Aus ihrer Kabine wird plötzlich ihr gesamter Schmuck entwendet. Der Täter kann das Schiff erst verlassen, wenn es im nächsten Hafen anlegt. Aufgabe des Spielers ist, den Dieb zu entlarven, bevor die Schiffsreise beendet ist.

Spielidee 3: Die geheimnisvolle Insel

Eine Flaschenpost mit einer Seekarte wird gefunden, auf der eine bisher unbekannte Insel eingezeichnet ist. Aufgabe des Spielers: zur Insel reisen und sie erforschen.

Diese drei Themen sind Beispiele, wie eine zündende Idee zu einem Spiel aussehen könnte. Es gibt unzählige spannende Abenteuerbücher. Sehen Sie sich zwanglos in Büchereien, Bibliotheken und Buchhandlungen um. Bestimmt finden Sie ein Buch, das Ihnen wertvolle Anregungen zu einem Spiel geben kann. Empfehlenswert sind die Fantasy-Buchreihen verschiedener Verlage. Weitere Vorschläge finden Sie in Kinofilmen – denken Sie an die verschiedenen Computerspiele, die erfolgreichen Film-Hits nachempfunden wurden, wie »Der dunkle Kristall« oder »Indiana Jones«. Betrachten wir die vorgestellten Themen genauer:

Der Löwe ist los

Auf den ersten Blick scheint in der Idee, »Spieler muß ausgerissenen Löwen fangen«, nicht allzuviel zu stecken.

Der Spielort: eine große Stadt. Damit das Spiel nicht zu monoton wird, setzen wir einige Parks oder Grünanlagen in die Abenteuerlandschaft »Stadt«. Um den Löwen zu fangen, benötigt der Spieler einen Käfig und frisches Fleisch, das den Löwen anlocken soll. Den Käfig kann er nur mit einem Lastwagen befördern, den der Spieler z.B. mieten muß. Der Lastwagen braucht ab und zu Diesel. Das kostet Geld – der Spieler hat aber keines. Vielleicht gibt es in der Stadt ein Spiel-Casino oder eine Bank, bei der sich der Spieler Geld leihen kann. Vorsicht, da sind ein paar Diebe, die auf das erworbene Geld scharf sind! Was passiert, wenn der Lastwagen unterwegs kaputtgeht?

Juwelenraub auf der »Queen Mary«

Widmen wir uns dem zweiten Beispiel. Der Spielort ist ein Schiff – mit Kabinen, Speisesaal, Aufenthaltsräumen, Decks usw. Die Handlung kann man mit der eines Kriminalfalls vergleichen. In dieser Art Adventure müssen viele Personen mitspielen, die alle Motive und Alibis haben könnten. Der Spieler beschattet die Personen und befragt sie. Er übernimmt die Rolle eines Detektivs. Während der Suche nach dem Täter kann ein Mordanschlag passieren (er muß aber nicht unbedingt gelingen) – ein Attentat auf eine Person (nicht auf den Spieler!), die den Täter kennt und nach dessen Ansicht zu viel weiß.

Ein Spiel dieser Art in ein Programm zu packen, ist zwar möglich, aber relativ schwierig. Das Programm muß alle Personen (den Spieler ausgenommen) steuern. Die Personen müssen sich bewegen, handeln und auf den Spieler eingehen (reagieren).

Die geheimnisvolle Insel

Das könnte ein typisches Abenteuerspiel sein. Durch eine Seekarte findet der Spieler den Weg zu einer unbekannten Insel. Auf der Insel leben für ausgestorben gehaltene Tiergattungen (Saurier) und Steinzeitmenschen. Auf der Insel gibt es Höhlensysteme, Flüsse, Dschungel, Berge, Seen und einen Vulkan. Die Insel birgt ein Geheimnis, das der Spieler entdecken muß. Das naheliegendste wäre, einen Schatz zu verstecken. Man kann dem Spiel auch einen Hauch Science-fiction verleihen: Der Spieler entdeckt im Inneren des Vulkans eine technisch hochentwickelte Apparatur – die Inselbewohner entpuppen sich als perfekte Roboter. Vielleicht ist die Insel nur eine Tarnung für eine geheime Raumschiff-Station? Anhand dieses Beispiels können Sie gut erkennen, wie man überraschende Effekte erzielt, durch die der Spieler, der eigentlich glaubt, er müsse nur einen vergrabenen Schatz finden, völlig verblüfft wird. Ich möchte ein Beispiel herausgreifen, an dem wir lernen wollen, welche Tabellen man erstellen muß und wie man eine Adventure-Karte zeichnet.

Einigen wir uns auf das dritte Beispiel – die geheimnisvolle Insel. Machen wir uns zunächst Gedanken über die Umgebung der Insel. Wie sieht es auf einer Südsee-Insel aus? Da ist der Strand – dort soll auch unser Spiel beginnen. Der Spieler rudert mit einem Boot vom Schiff zum Ufer. Von dort kommt er in einen tropischen Regenwald oder Dschungel. Auf unserer Fantasie-Insel erhebt sich ein hohes Gebirge, in dessen Zentrum ein riesiger Vulkan aufragt. Mehrere Flüsse durchqueren das Eiland, ihre Quellen liegen im Gebirge. Wo Flüsse und Vulkane sind, existieren auch heiße Quellen und Lagunen. Die Insel ist bewohnt. Wir stoßen auf ein kleines Dorf mit Eingeborenen oder auf mehrere mit unterschiedlichen Stämmen. Ab diesem Punkt benötigen wir eine Skizze. Malen wir unseren Lageplan, ähnlich wie in Bild 1.

Diese Grafik hat wenig mit einer richtigen Adventure-Spielkarte gemeinsam. Die Größenverhältnisse auf dem Bild stimmen nicht! Es soll lediglich ein grober Überblick von den geplanten Adventure-Landschaften sein. Bei der vorgestellten Skizze der Insel sind folgende Interpretationen möglich: Ein riesiges Gebirge erstreckt sich über den westlichen Teil der Insel. Im Zentrum des Gebirges steht ein Vulkan. Im Gebirge entspringen mehrere Flüsse, die die Insel durchfließen. Einer der Flüsse mündet in einen großen See im östlichen Teil der Insel. An einem der Flüsse liegt ein Eingeborenendorf. Im nordwestlichen Teil, am Fuß des Gebirges, befindet sich eine riesige Mauer. Hinter der Mauer liegt ein Tempel versteckt. Der restliche Teil der Insel besteht aus Wäldern und Sümpfen.

Damit gewinnen Sie einen bildlichen Eindruck der Gegend, in der das Adventure spielen soll. Während Sie die Skizze betrachten, kommen Ihnen schon neue Ideen und Anregungen über den Verlauf des Adventures.

Worüber wir uns bisher keine Gedanken gemacht haben, sind die Aufgaben, die der Spieler lösen muß. Der nächste Schritt besteht darin, die eigentliche Spielkarte zu entwerfen, mit deren Hilfe wir das Spiel programmieren werden.

Ein großes Problem muß noch gelöst werden: Hat man die Spielkarte fertig ausgearbeitet, ist es schwierig, spätere Änderungen vorzunehmen. Während man das Spiel program-



miert, kommen einem viele Ideen in puncto Spielwitz, die man mit einbauen will. Zeichnen Sie deshalb die Spielkarte sorgfältig und sauber, damit Sie die Übersicht nie verlieren.

Entwurf der Spielkarte

Die Gedankengänge beim Erstellen einer Spielkarte müssen exakt die Umkehrung derjenigen sein, mit denen Sie das Adventure lösen.

Bestimmt haben Sie schon ein Adventure gespielt und sich dabei Notizen über Räume und Objekte gemacht. Der Computer liefert zu Programmbeginn seinen ersten Lagebericht, z.B.:

```
SIE BEFINDEN SICH IN EINEM GROSSEN ZIMMER
IM NORDEN BEFINDET SICH EINE TUER, DIE OFFEN IST
MOEGLICHE RICHTUNGEN: S, N, O
```

Der gewissenhafte Abenteurer macht seine erste Notiz. Er zeichnet ein Kästchen und beschreibt kurz den Raum (Bild 2). Dann malt er Pfeile in die möglichen Richtungen. Der Pfeil zeigt in die Richtung, durch die man den Raum verlassen kann. Geht man z.B. nach Norden, weiß man noch nicht, ob man anschließend durch Bewegung nach Süden wieder zum Ausgangsort zurückkehren kann. Deshalb nur ein Pfeil. Der nächste Schritt besteht darin, die einzelnen Richtungen auszuprobieren. Gehen wir nach Osten.

Das Programm liefert daraufhin folgenden Lagebericht:

```
SIE BEFINDEN SICH IN EINEM LANGEN GANG
AM ENDE DES GANGES IST EIN KLEINES FENSTER
MOEGLICHE RICHTUNGEN: W, O
```

Wir können den Lageplan ergänzen. Aus der »Lageberichtserstattung« des Programms geht hervor, daß man nach Westen oder Osten laufen kann (Bild 3).

Logischerweise müßte man wieder in das große Zimmer gelangen, wenn man zurück nach Westen läuft. Probieren wir es aus. Wir gehen zurück in westliche Richtung – und sind wieder im großen Zimmer. Jetzt können wir den Pfeil in der entgegengesetzten Richtung mit einer Spitze versehen. (Bild 4).

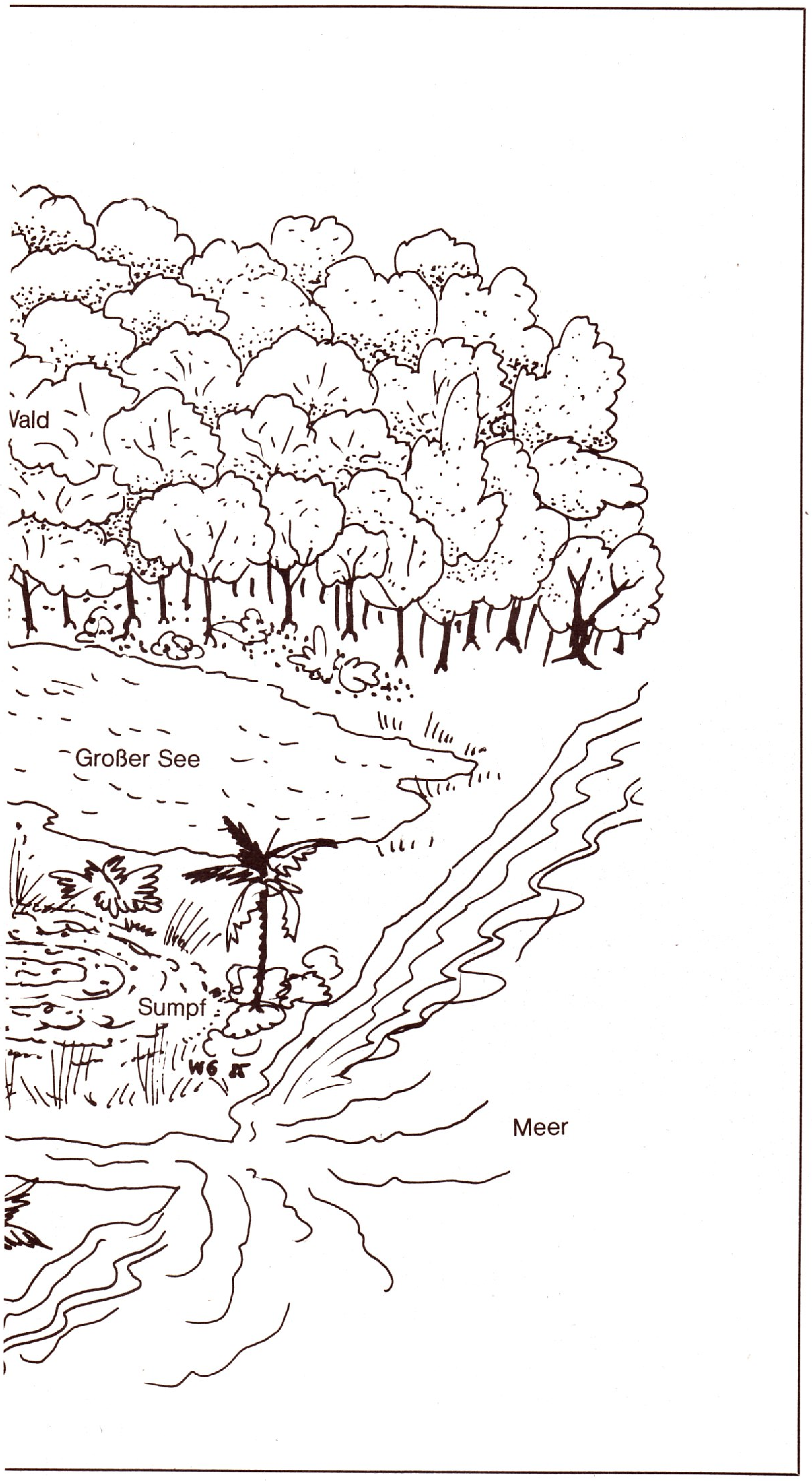


Bild 1. Die Skizze der Schatzinsel, im Fachjargon »Scribble« genannt

Es scheint viele Adventure-Programmierer zu geben, denen es eine wahre Freude bereitet, den Spieler durch unlogische Pläne zu verwirren. Auch in professionellen Adventures (z.B. der Klassiker »Hobbit« von 1984) wird ein derartiges Verwirrspiel getrieben. Es kann sehr frustrierend sein, wenn der Spielplan völlig unlogisch aufgebaut ist. Deshalb auch mein Rat an alle angehenden Adventure-Programmierer: Vermeiden Sie so etwas in Ihren Abenteuerspielen. Der Spieler sollte durch derartige Fehler nicht vom Spiel abgelenkt werden, außer es ist vom Programmierer so vorgesehen.

Einige Beispiele:
- Man stürzt in eine Grube.
- Ein Fallgitter fällt herunter und versperrt den Ausgang.
- Hinter dem Spieler fällt eine Tür zu, die sich nicht mehr öffnen läßt.

Eine Szene, bei der ein Fallgitter herunterfällt, würde man wie in Bild 5 kennzeichnen. Sie müssen beim Pfeil angeben, warum eine Umkehr unmöglich ist. Dazu genügt ein markantes Wort (z.B. Fallgitter). Oft befinden sich Türen zwischen einzelnen Räumen. Diese Türen werden im Plan skizziert (Bild 6). Sie werden durch gestrichelte Linien mit der entsprechenden Bemerkung (Schlüssel) dargestellt. Im Fall einer magischen Tür würde sie eventuell »Zauberspruch«, bei einem elektronisch gesicherten Durchgang vielleicht »Identi-

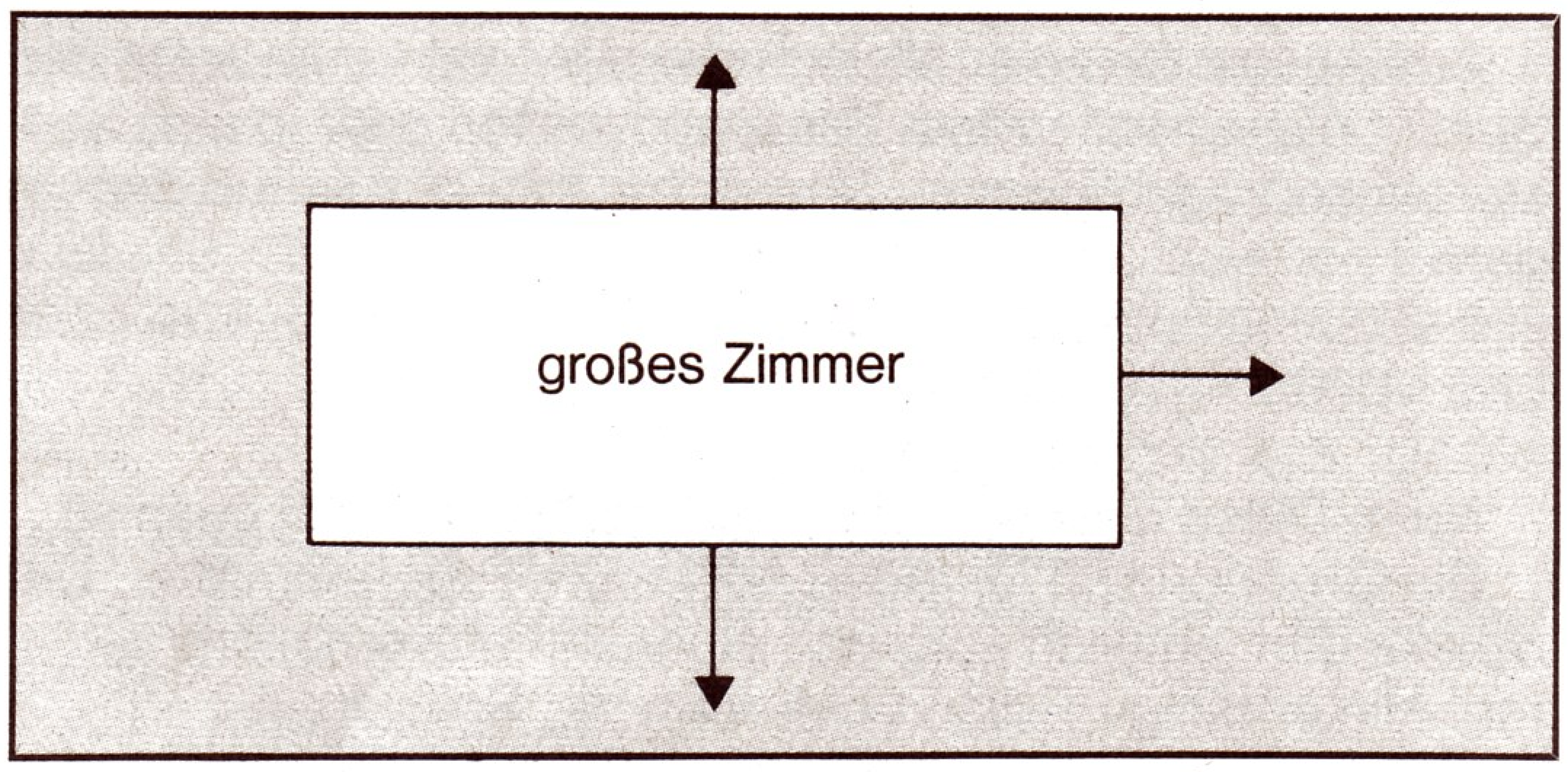


Bild 2. Der Raum besitzt drei Ausgänge

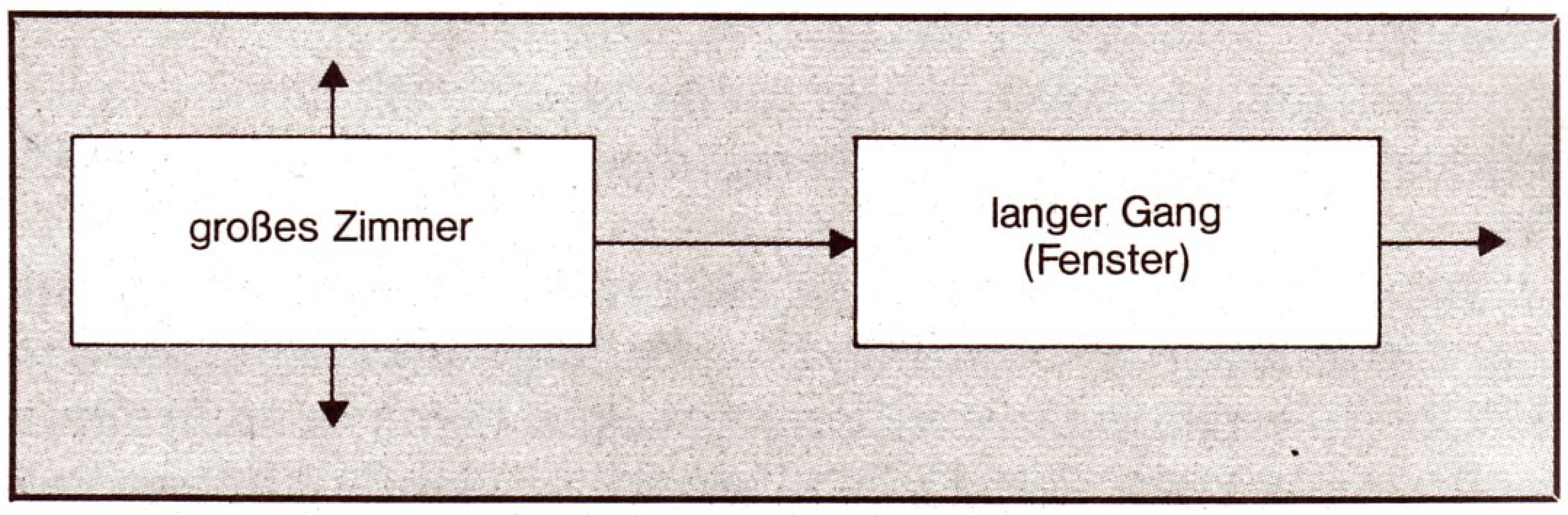


Bild 3. Der Weg führt durch einen langen Gang nach Osten

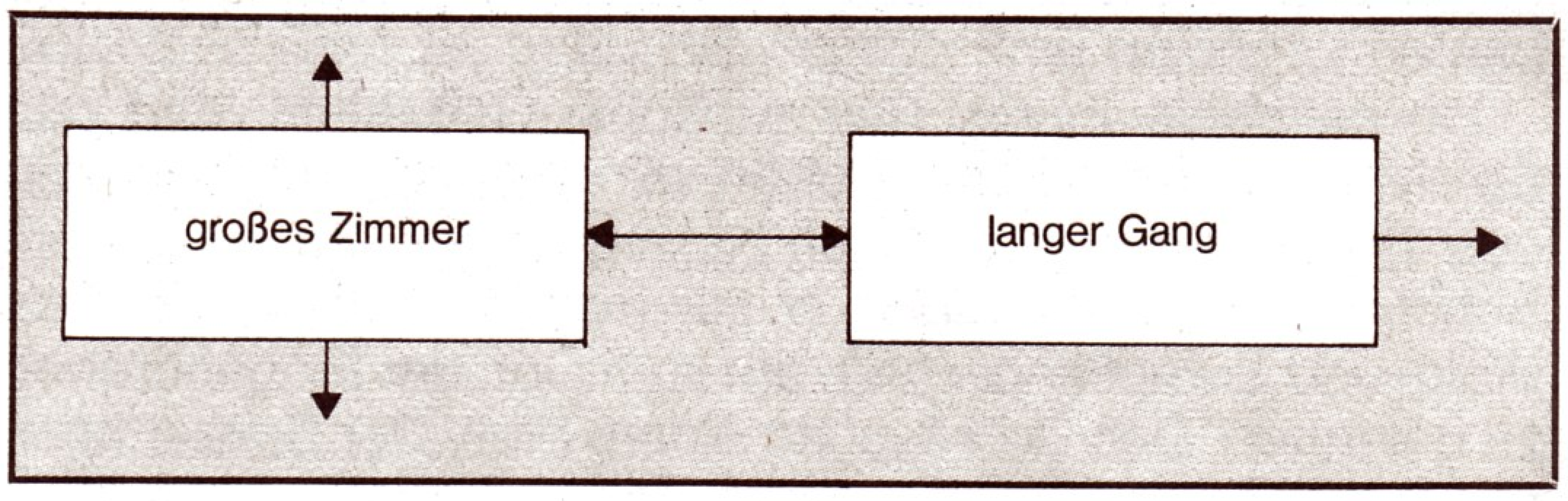


Bild 4. Freier Rückzug – der Pfeil mit zwei Spitzen macht es möglich

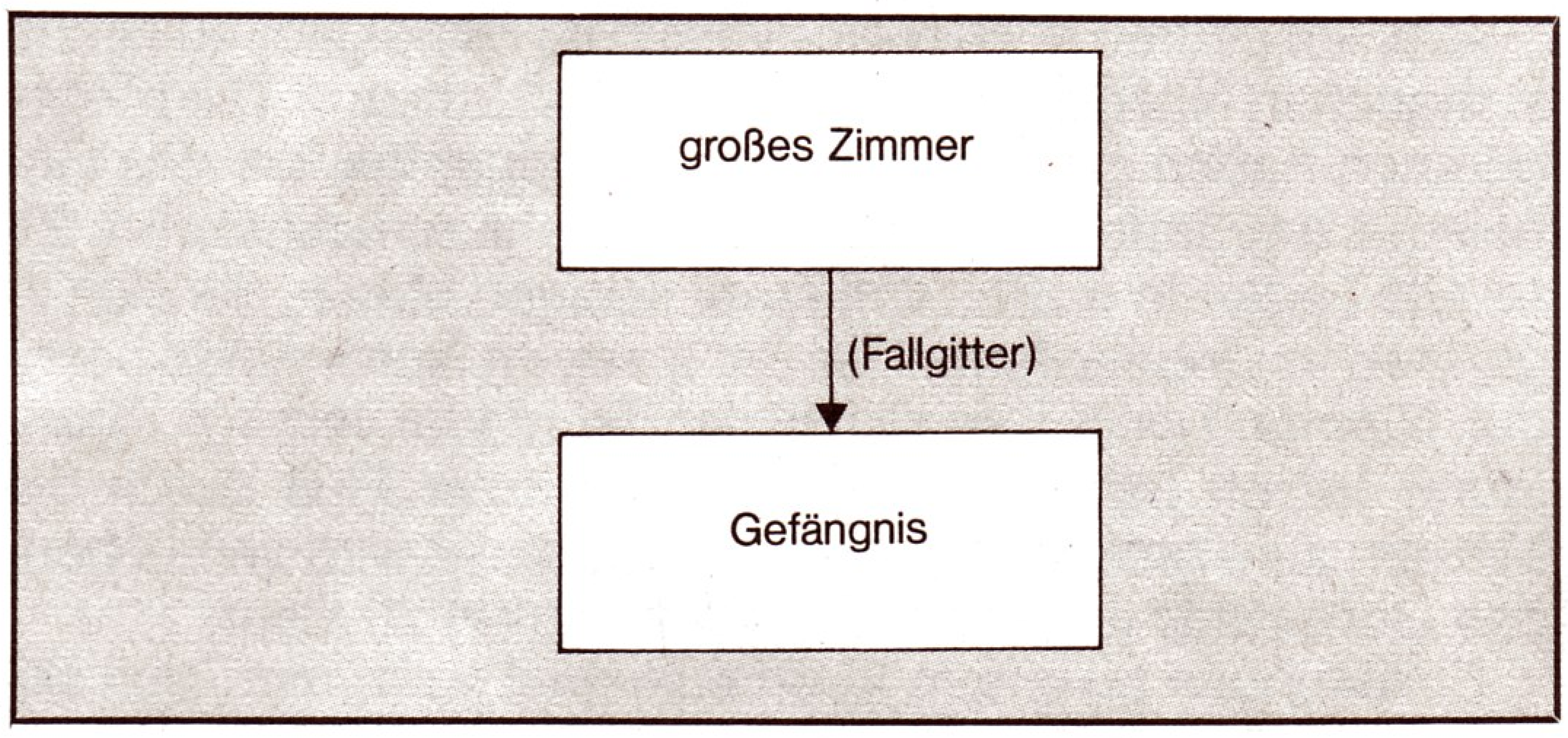


Bild 5. Beispiel einer ausweglosen Situation

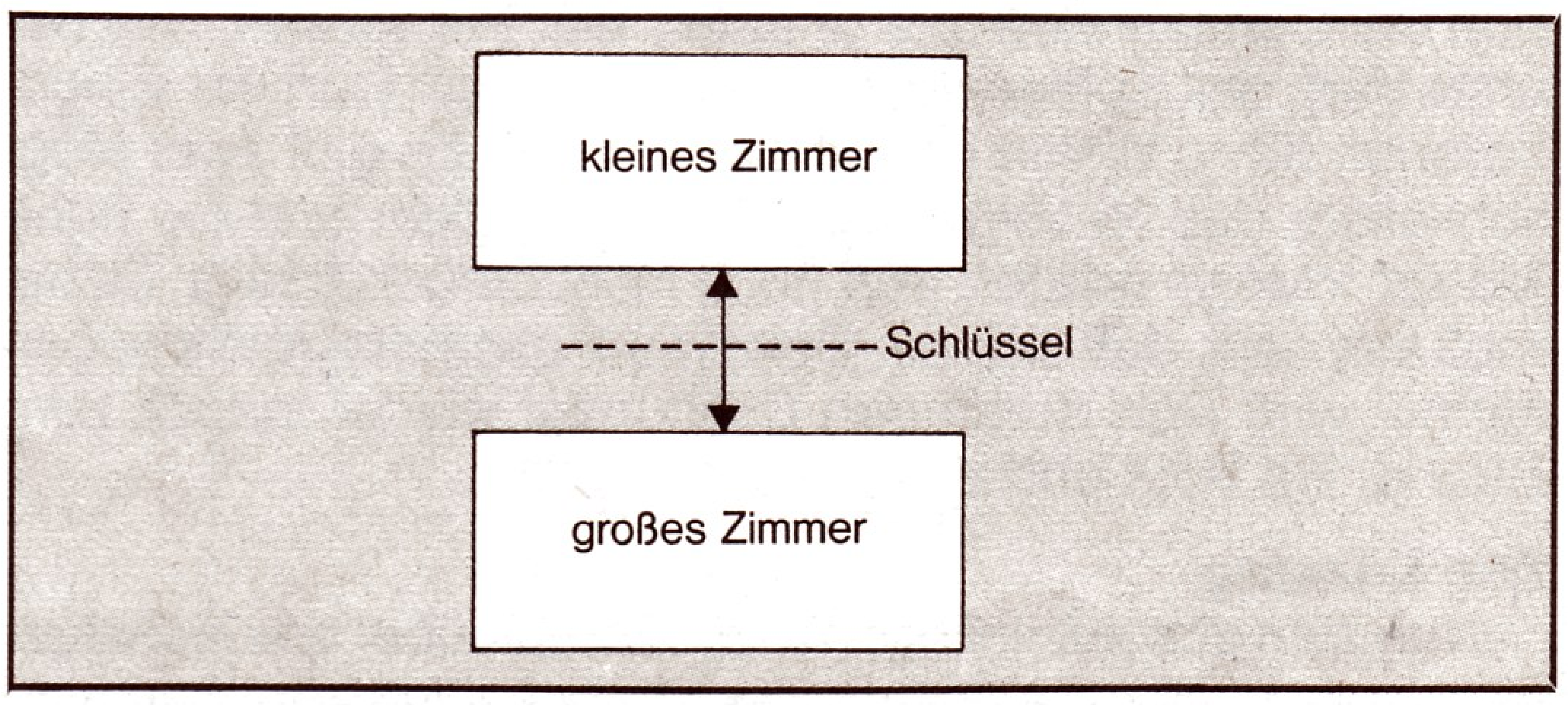


Bild 6. Ohne Schlüssel führt kein Weg ins kleine Zimmer

tätskarte« lauten. Fehlt diese Bemerkung, bedeutet dies, daß man die Tür ohne weiteres passieren kann.

Eine andere Spielsituation: Der Spieler stürzt in eine Grube, die er mit Hilfe eines Seiles wieder verlassen kann. Die entsprechende Skizze sehen Sie in Bild 7.

Läuft man demnach vom großen Zimmer in südlicher Richtung, stürzt man in eine Fallgrube. Sie kann nur mit einem Seil wieder verlassen werden. Die Bedingung »Seil« finden Sie in dem großen Pfeil, der von der Grube zum großen Zimmer führt.

Aus diesem Beispiel läßt sich etwas lernen – man sollte den Spieler nie vor vollendete Tatsachen stellen oder in eine aussichtslose Lage bringen. Zahlreiche Adventures tun das zu gern:

```
SIE SIND IN EINE FALLGRUBE GESTUERZT UND HABEN SICH
DAS GENICK GEBROCHEN!
WOLLEN SIE NOCHMAL SPIELEN?
```

Nein! Diese Spiele werden von mir nur einmal gespielt.

Ein Grundprinzip für jeden Adventure-Programmierer: Der Spieler darf lediglich aufgrund eigenen Fehlverhaltens verlieren. Hinterhältige Fallen verderben die Spiellaune.

Mehr als vier Richtungen

Scheuen Sie sich nicht, zehn Himmelsrichtungen zu verwenden:

```
N, S, O, W, NO, NW, SO, SW, RAUF, RUNTER
```

dadurch lassen sich bedeutend interessantere Pläne erstellen.

Die Richtungen N, S, O, W, NO, NW, SO, SW werden auf der Lagekarte mit einfachen Pfeilen dargestellt, »RAUF« und »RUNTER« hingegen mit Doppelpfeilen, damit keine Irrtümer auftreten können (Bild 8). Die Bedeutung der Skizzensymbole sollte uns jetzt geläufig sein.

Wie setzen wir die Skizze der Insel in einen sinnvoll aufgebauten Lageplan um?

Ich empfehle Ihnen, sich zunächst Gedanken über den Spielverlauf zu machen. Gehen wir von folgender Situation aus: Der Spieler befindet sich zu Beginn des Spiels auf dem Schiff. Von dort aus rudert er zur Insel. Er durchquert eine Wald- und Sumpflandschaft, um zum Dorf der Eingeborenen zu gelangen. Das Dorf liegt an einem Fluß. Der Spieler kann das Dorf mit seinem Ruderboot vom Meer aus über den Fluß nicht erreichen. Die Strömung ist zu stark. Im Eingeborenendorf erfährt der Spieler ein Geheimnis:

»Der vor Jahren erloschene Vulkan im Nebelgebirge strahlt in manchen Nächten ein merkwürdiges Licht aus. Die Eingeborenen haben große Angst davor. Neugierige, die das Geheimnis ergründen wollten, sind nie mehr zurückgekehrt. Eine alte Legende erzählt von einer riesigen Mauer, die sich irgendwo im unbewohnten Teil der Insel, im Nordwesten befindet. Niemand weiß, was sich hinter der Mauer verbirgt. Abenteurer haben berichtet, daß ein schwarzer Fluß durch die Mauern zum großen See fließt.« Der Spieler wird versuchen, das Geheimnis (und damit das Adventure) zu lösen.

Einleitungen zu einem Abenteuerspiel sollte man sich angewöhnen, da sie das Interesse am Spiel steigern. Sie dürfen nur oberflächliche Informationen enthalten. Der Spieler kennt nun die Geheimnisse, die er lüften will. Wir wissen, daß der erste Weg des Spielers vom Inselstrand zum Dorf der Eingeborenen führt. Wohin soll er jetzt gehen? Es gibt zwei Möglichkeiten: das Nebelgebirge mit dem Vulkan und die hohe Mauer. Irrwege werden jetzt noch nicht berücksichtigt. Dabei muß klar sein, daß sich der Spieler auf der Insel frei bewegen kann – vom Dorf durch den Dschungel wieder zurück zum Strand, ins Boot und zurück zum Schiff.

Die Lösung

Nun zum absolut richtigen Weg. Er führt über das Nebelgebirge zum Vulkan. In seinem Inneren befindet sich eine hochentwickelte, vollautomatisierte Fabrik. Sie wird von Robotersklaven betrieben. Gesteuert werden sie durch einen wahnsinnigen Professor, der die Weltherrschaft an sich reißen will.

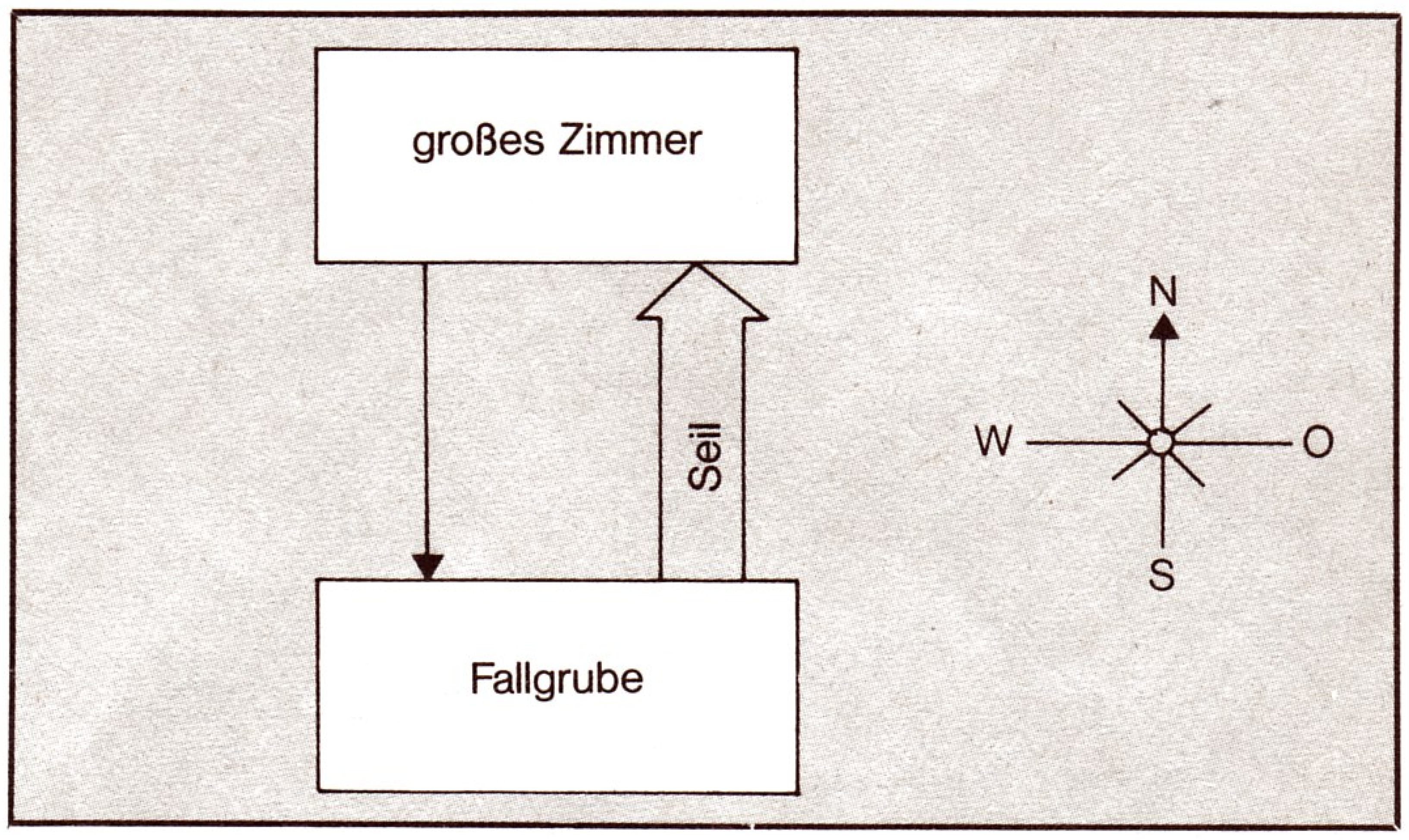


Bild 7. Ein Seil hilft Ihnen, die Fallgrube zu verlassen

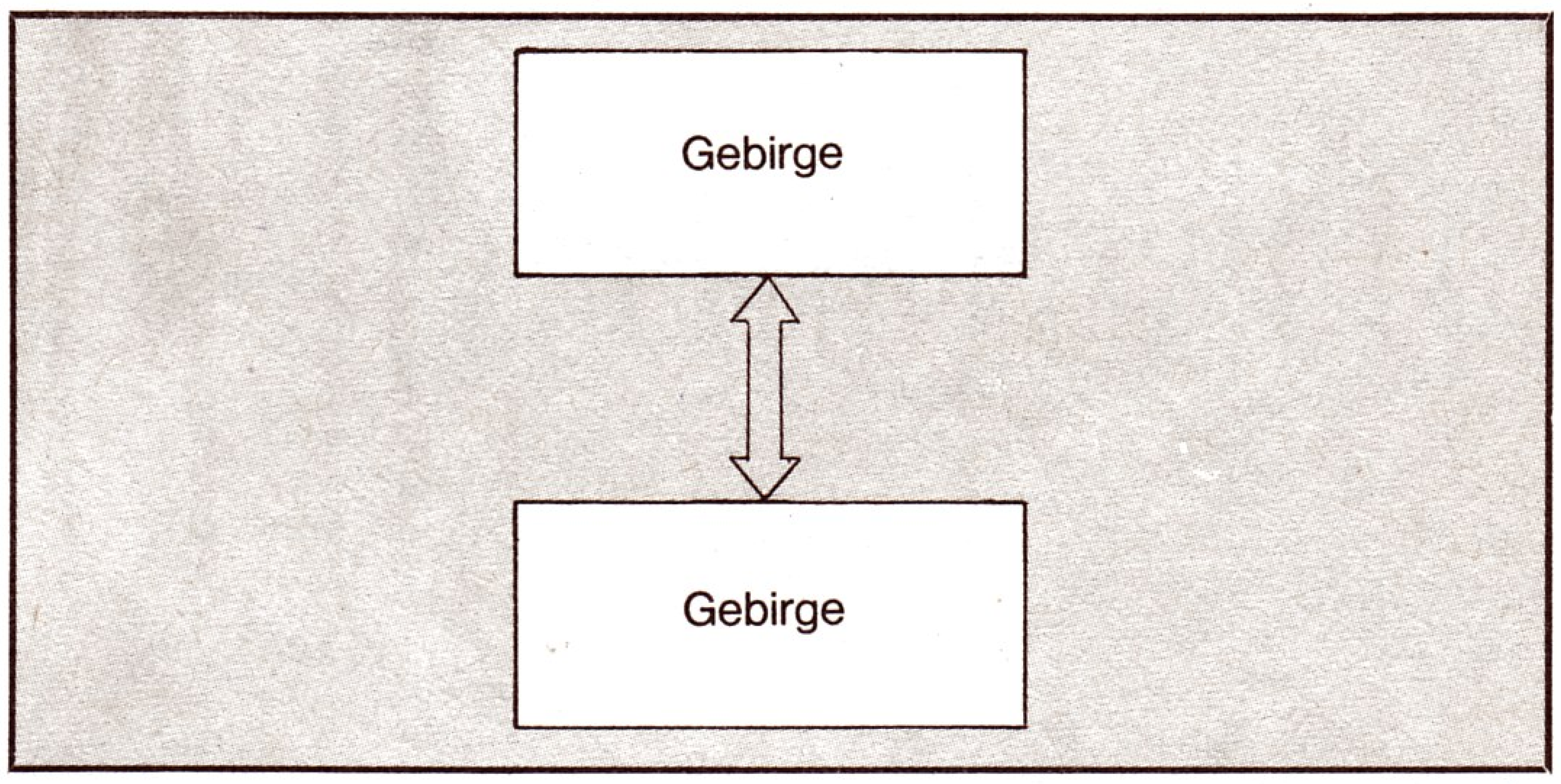


Bild 8. Vom Fuß der Berge bis zur Spitze: aufwärts oder abwärts, beide Richtungen sind möglich

In der Fabrik durchlebt die Spielfigur einige kribbelnde Situationen, bis sie einen Geheimgang entdeckt. Er führt unter dem Nebelgebirge zum Tempel hinter der Mauer. Der Spieler betritt den Tempel und muß dort einige »Nüsse« knacken. Anschließend kommt er über den schwarzen Fluß unter der Mauer hindurch zum großen See, von dort zum Strand und zurück zum Schiff – das Adventure ist gelöst.

Grundplankarte und Spielplan

Es ist an der Zeit, eine neue Skizze zu zeichnen, zumindest unsere alte Inselskizze ein wenig zu überholen. Sie wird in die Grundplankarte nach dem vorgestellten Kästchen-und-Pfeil-Prinzip umgesetzt (Bild 9).

In der Landkarte sind fünf große Bereiche und ihre Verbindung zueinander eingezeichnet: Urwald, Nebelgebirge, Vulkan, Tempel und der große See. Die Hauptgebiete der Insel wurden in einzelne Blöcke aufgeteilt. Nach dieser Aufteilung die eigentliche Landkarte anzufertigen, ist bedeutend leichter als nach der Skizze von Bild 1.

Der Strand und das Schiff wurden ebenfalls als Blöcke eingezeichnet. Je größer ein Block ist, desto mehr Räume kann er enthalten. Zwischen den einzelnen Bereichen finden Sie einfache und doppelte Pfeile, die bedingungsabhängige Verbindung zwischen den Räumen. Leicht können Sie daraus ersehen, daß man lediglich über den Geheimgang vom Vulkan zum Tempel kommt. Von dort geht es nur über den Fluß zum großen See.

Der nächste Schritt besteht darin, die einzelnen Gebietsblöcke in einzelne Räume zu gliedern. Es gibt keine verbindlichen Vorschriften darüber, wieviele Räume und wieviel Action man in ein gutes Adventure packen sollte. Als Faustregel könnte gelten: eine Relation zwischen Raumanzahl und Action von 1:1.

Der Spielplan (Bild 10) liegt nahezu fertig vor uns. Aus ihm ist ersichtlich, welche Räume existieren und wie sie miteinander verbunden sind. Dieser Plan ist knapp gehalten und nicht ausreichend modifiziert. Einige Beispiele dazu: Man kann

den Wald vergrößern. Der Spieler könnte durch den Wald zur Mauer kommen. Dort kommt man zwar nicht weiter, denn der Tempel ist nur durch den unterirdischen Geheimgang vom Gebirge aus zu erreichen. Aber in der Vorgeschichte des Spiels haben wir die sagenumwobene Mauer erwähnt, deshalb muß sie im Spiel auftauchen. Außerdem sollte man die unterirdische Fabrik, die gesamten Wandermöglichkeiten im Gebirge und den Tempelplan mit einbauen.

Langeweile vermeiden

Es wird Zeit, sich Gedanken über den Aktionsverlauf des Spiels zu machen.

Tatsächlich arbeitet man beim Erstellen von Adventure-Spielen immer an mehreren Dingen gleichzeitig.

Während man die Landkarte zeichnet, kommen einem neue Ideen. In Raum 11 finden Sie z.B. eine Höhle. Von ihr war noch nie die Rede. Wie kommt sie dann in den Plan? Ganz einfach: Die Idee kam mir beim Zeichnen der Landkarte für das Gebirge. Wie wird ein Spieler reagieren, wenn er die Lagebeschreibung erhält:

```
SIE BEFINDEN SICH IM GEBIRGE.
SIE SEHEN EINEN HOEHLENEINGANG!
```

Jeden Spieler wird die Neugier plagen. Er geht in die Höhle. Ist sie leer, wäre das langweilig. Legen Sie in die Höhle einen Gegenstand, der für das weitere Spiel sehr wichtig ist. Das könnte ein Schlüssel, ein Schwert oder ähnliches sein. Damit ist gewährleistet, daß der Spieler in die Höhle gehen muß – früher oder später.

Bei einem Adventure sollte man nur dann etwas bekommen, wenn man es sich verdient hat. Der Spieler darf nicht einfach in die Höhle gehen, den Gegenstand nehmen und sie wieder verlassen. Wir müssen ihm eine Aufgabe stellen.

Eine abgedroschene Lösung wäre, den Gegenstand im Boden der Höhle zu vergraben (der Spieler braucht dazu eine Schaufel) oder zu verstecken. Mehr Spielfreude läßt eine schwierige Aufgabe aufkommen, bei der der Spieler nur durch Nachdenken weiterkommen oder überleben kann. Da kommt mir ein Gedanke: Ein wilder Bär. In der Höhle findet der Spieler einen interessanten Gegenstand. Er nimmt ihn an sich. Plötzlich tönt ein gefährliches Brummen vom Höhleneingang her. Ein riesiger Bär betritt mit Gebrüll die Höhle und nähert sich dem Spieler. Die Frage ist: Wie kommt er lebend aus der Höhle heraus? Wegrennen ist zu leicht. Den Bär mit einem Schwert zu bekämpfen ist nicht möglich, da er zu stark ist.

Wir haben vorher klargestellt, daß man in einem guten Adventure den Spieler nie vor vollendete Tatsachen stellen darf. Die rettende Idee: Bären haben Angst vor Feuer. Der Spieler muß schnell etwas Holz vom Boden nehmen (das dort liegen muß) und es mit seinem Feuerzeug (das er hoffentlich in seiner Ausrüstung hat) entzünden. Der Bär bekommt Angst und flieht!

Viel »Action« in Adventures packen!

Das war ein Beispiel, wie eine Action-Szene aussehen könnte.

Die Arbeitsschritte dazu:

- Ausdenken einer Gefahrensituation, auch wenn sie zunächst ausweglos erscheint.
- Bei welchen Reaktionen verliert der Spieler?
- Welche Möglichkeiten hat er, die Situation zu überstehen? Kann der Spieler Gegenstände verwenden, die sich im Raum befinden? Gibt es Fluchtwege? Hilft ein Zauberwort?

Bedenken Sie: Je intensiver Sie eine Spielsituation durchdenken, je schwieriger eine vernünftige Lösung zu finden ist, desto interessanter und anspruchsvoller wird das Spiel für den »Abenteurer« sein. Betonen möchte ich den Ausdruck »vernünftig«. Lösungen sollten der Realität entsprechen.

Nach dem Einbau von Action-Szenen muß zumeist die Spielkarte (Bild 10) geändert werden.

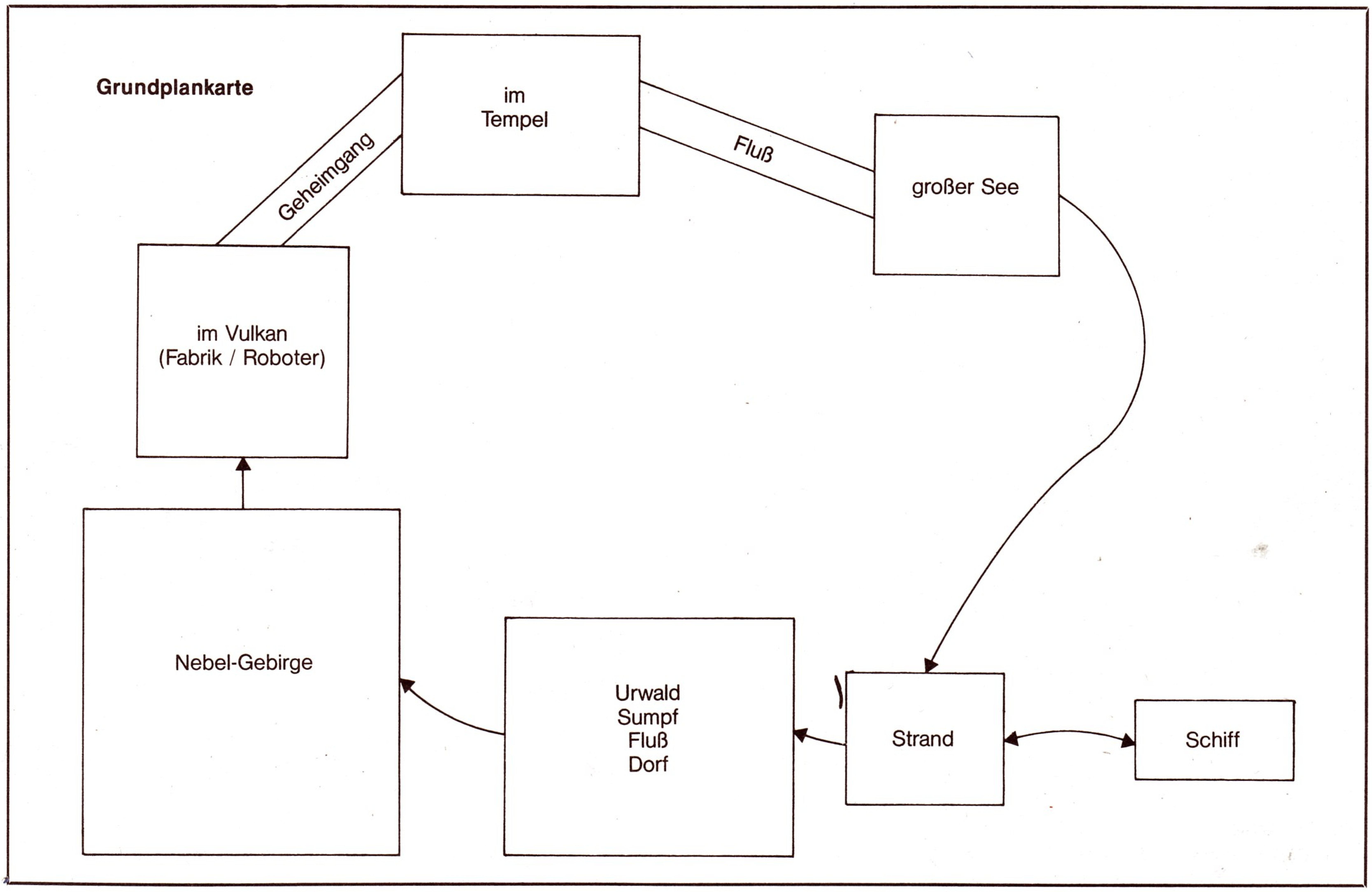


Bild 9. Diese Grundplankarte wurde aus der Skizze entwickelt

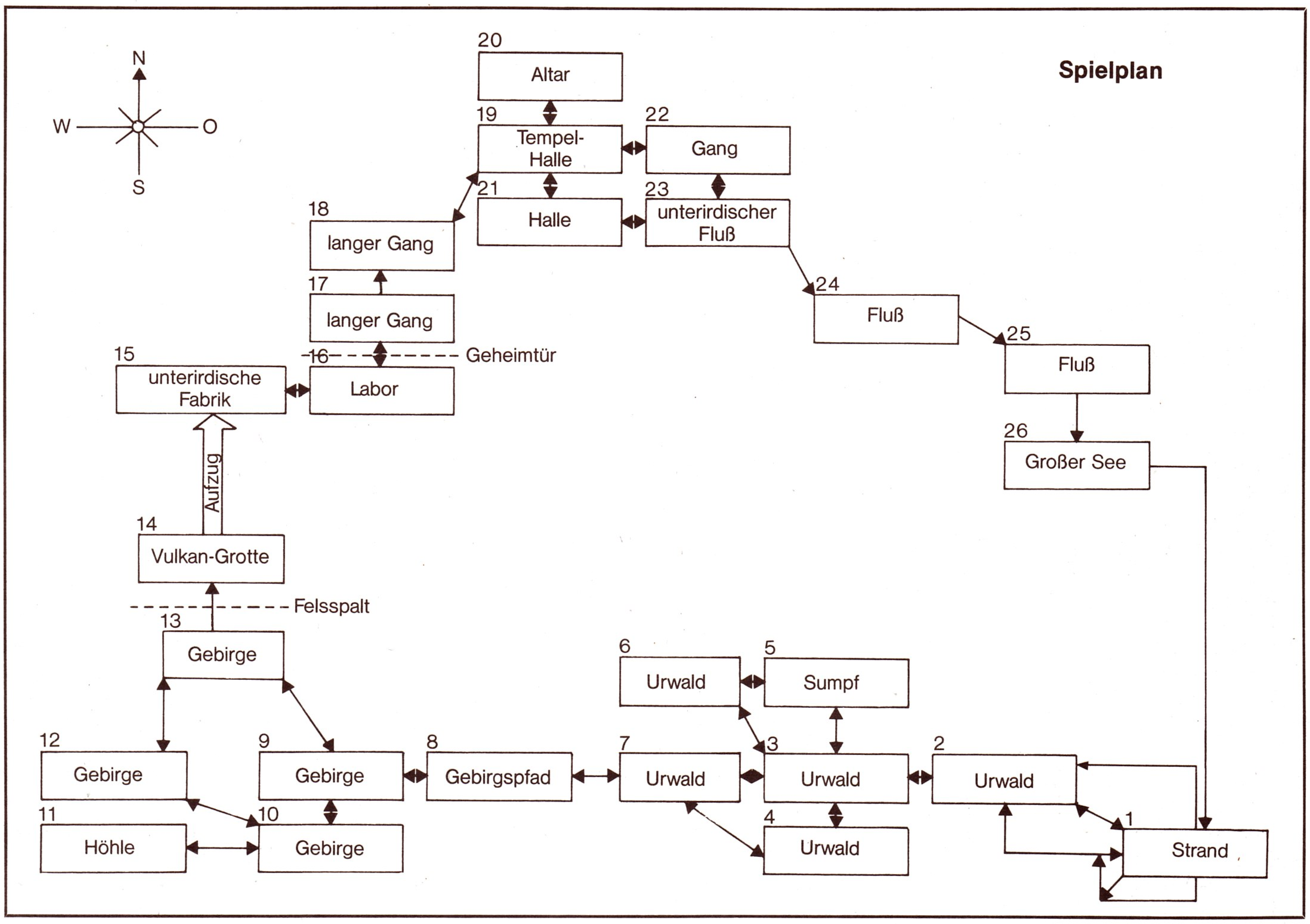


Bild 10. Genaue Übersicht der Raumverteilung: der Spielplan

Richtungen definieren

Von Raum 1 (Strand) kann man in mehrere Richtungen gehen. Sie führen in den Urwald. Betrachten Sie die Räume 4 und 7 im Wald und ihre Verbindungen zueinander. Sie werden feststellen, daß man von Raum 7 nach Südosten gehen muß, um in Raum 4 zu gelangen. Es ist sehr wichtig, die Himmelsrichtungen NO, SO, SW, NW ebenfalls in der Karte zu vermerken. Es bedeutet zwar mehr Programmieraufwand als bei vier Himmelsrichtungen, aber der Spielplan wird mit acht bzw. zehn Richtungsmöglichkeiten (zusätzlich »oben« und »unten«) komplexer. Das zwingt den Spieler, sich Aufzeichnungen während des Spiels zu machen.

Zwischen den Räumen 13 und 14 sehen Sie eine gestrichelte Linie (Felsspalt). Sie bedeutet, daß der Spieler nicht auf direktem Weg von Raum 13 in Raum 14 kommt, sondern nur durch einen Felsspalt, den er finden muß. Die Spalte könnte durch einen Felsblock versperrt sein, den man erst wegrollen muß. Betrachten Sie den Felsspalt als Tür zwischen Raum 13 und 14. Auch zwischen Raum 16 und 17 befindet sich eine gestrichelte Linie (Geheimtür). Raum 14 und 15 verbindet ein Doppelpfeil (Aufzug). In Raum 14 (Grotte) müssen Sie den Lift

Die Programmiertechnik

besteigen, um in Raum 15 zu kommen. Zwischen den Räumen 24 und 25 (Fluß) sehen Sie einen Pfeil, der lediglich in eine Richtung weist. Der Spieler kann sich nur in der Richtung fortbewegen, in die der Fluß treibt. Da der Fluß obendrein reißend ist und tückische Stromschnellen birgt, ist der Spieler nicht in der Lage, gegen die Strömung zu rudern. Der Sinn des Numerierens der Räume sollte damit klar geworden sein: Sie können sich leichter Notizen zu den einzelnen Räumen machen.

Dazu sollten Sie eine Tabelle führen, wo sich die einzelnen Gegenstände befinden. Sie können sie auch direkt in die Karte eintragen. Wichtig ist, daß Sie beim Programmieren genügend Notizen haben, auf die Sie zurückgreifen können. Ziel dieses Kurses ist, Ihnen das notwendige »Know-how« zum Programmieren anspruchsvoller Adventures zu vermitteln. Die einzelnen Programm-Module werden in Funktion und Aufbau ausführlich dokumentiert. Sie finden diese Muster-Programme unter den Namen »LISTING 1« bis »LISTING 26« auf der beiliegenden Diskette. Basic-Kenntnisse sind unbedingt erforderlich, denn dies ist kein Grundlagenkurs für diese Programmiersprache. Auf Kniffe und Programmiertricks werden wir ausführlich eingehen.

Programmiermethoden

Den Überblick über ein langes Basic-Programm kann man gewährleisten, wenn man es gut strukturiert. Solche Programme sind in der Regel länger als normale. Wir müssen eine Zwischenlösung finden:

- Das Programm wird in kleine Teilprogramme (Module) zerlegt.
- Die Module werden ausführlich in ihrer Funktionsweise erklärt. Jedes Modul ist in sich abgeschlossen. Es kann individuell für jedes Programm verwendet werden. Das Modul wird im Normalfall mit »GOSUB« aufgerufen.
- Die einzelnen Modul-Teile werden vom Hauptprogramm aufgerufen. Das Hauptmodul ist die Steuerungszentrale.
- Verwenden Sie für die Module reservierte Zeilennummern, z.B Befehlseingabemodul von Zeile 50000 bis maximal Zeile 50999 usw.

Die Module werden in den folgenden Abschnitten ausführlich behandelt. Ein Abschnitt baut auf dem anderen (vorhergehenden) auf. Laden Sie das entsprechende Listing-Modul von der beiliegenden Diskette in den C64 und listen es Programmzeile für Programmzeile. Verfolgen Sie die Erläuterung der einzelnen Programmteile. Es ist wichtig, daß Sie die Funktionsweise der Module verstehen. Sind Sie damit vertraut, können Sie damit experimentieren.

Befehlseingabe

Gerade für die Programmierung von Abenteuerspielen ist es zwingend notwendig, mit der Technik der Stringverarbeitung vertraut zu sein. Obwohl die eigentlichen Stringfunktionen in Basic leicht verständlich sind, haben Anfänger oft erhebliche Schwierigkeiten, sie sinnvoll einzusetzen. Vor allem dann, wenn mehrere Funktionen verschachtelt werden müssen. Der C64 kann folgende Strings in Basic verarbeiten:

ASC(X$)

Die ASC-Funktion liefert den Zahlenwert des ersten Zeichen eines definierten Strings.

```
PRINT ASC("A")
65
```

CHR$(X)

Das ist die Umkehrung der ASC-Funktion. CHR$(X) ergibt das Zeichen, das dem Zahlenwert X zugeordnet ist.

```
PRINT CHR$(65)
A
```

LEN(X$)

Diese Funktion ergibt die Anzahl der Zeichen, die ein String enthält.

```
PRINT LEN("ABENTEUER")
9
```

Achtung: Auch Leerzeichen werden im String mitgezählt.

LEFT$(X$,X)

Hier werden die linken X Zeichen von X$ als String berücksichtigt.

```
X$ = "ABENTEUER"
PRINT LEFTS(X$,5)
ABENT
```

RIGHT$(X$,X)

Diese Funktion ergibt die rechten X Zeichen von X$.

MID$(X$,S,X)

Diese Funktion ergibt X Zeichen von X$, beginnend mit dem Zeichen an Position S.

```
X$ = "ROTGELBGRUEN"
PRINT MID$(X$,4,4)
GELB
```

Wenn Sie den dritten Parameter weglassen, ergibt MID$ (X$,S) alle Zeichen ab Position S (»GELBGRUEN«).

STR$(X)

Damit haben wir die Möglichkeit, eine Zahl in einen String zu verwandeln.

```
X = 100+20+3
X$ = STR$(X)
123
```

VAL(X$)

Diese Funktion ist die Umkehrung der STR$-Funktion.

```
VAL(" 100" ) ergibt 100
VAL(" 123ABC" ) ergibt 123
VAL(" ABC123" ) ergibt 0, da das erste
Zeichen keine Zahl ist.
```

Anwendungsbeipiele

1. Prüfen, ob ein String in einem anderen enthalten ist.

Eine solche Routine ist besonders dann nützlich, wenn man dem Spieler ermöglichen will, Befehle abzukürzen (z.B UNT statt UNTERSUCHE).

Beispiel: Es soll überprüft werden, ob B$ in A$ enthalten ist.

```
10 A$ = "UNTERSUCHE"
20 B$ = "UNT"
30 IF B$=LEFT$(A$,LEN(B$)) THEN
 PRINT"B$ KOMMT IN A$ VOR!"
40 END
```

2. Ausdruck einer Zahlenkette.

Geben Sie folgendes Programm ein:

```
10 FOR I=1 TO 5
30 PRINT I;",";
40 NEXTI
50 END
```

Wenn Sie mit RUN starten, wird auf dem Bildschirm folgendes ausgedruckt:

```
1 , 2 , 3 , 4 , 5 ,
```

Die Zwischenräume der einzelnen Zeichen stören. Um dies zu vermeiden, machen wir in dem Programm den Umweg über den STR$(X)-Befehl.

Ergänzen wir das Programm:

```
20 X$=MID$(STR$(I),2)
```

und ändern Zeile 30:

```
30 PRINT X$;",";
```

Das Modul »Befehlseingabe« stellt einen Programmteil dar, bei dem der Computer einen Befehl vom Spieler erwartet. Bei vielen Adventures erscheint auf dem Bildschirm die Frage:

```
WAS NUN?
```

Unser Ziel ist, ein Befehlseingabe-Modul zu programmieren, das später universell eingesetzt werden kann.

Ich habe bereits erwähnt, daß jedes Modul dieselben Zeilennummern benutzen sollte. Bei längeren Programmen kann dadurch nie der Überblick verlorengehen. Das Befehlseingabe-Modul werden wir in den Zeilen 50000 bis maximal 50999 unterbringen. Aufgerufen wird es im Programm mit

```
GOSUB 50000
```

Laden Sie »LISTING 1« von der beiliegenden Diskette und starten es mit RUN. Der Computer fragt »WAS NUN?« und wartet auf eine Befehlseingabe, z.B.»NIMM SCHWERT«. Die Eingabe muß mit <RETURN> abgeschlossen werden.

Lassen Sie sich das Programm-Modul mit LIST anzeigen. Hier eine Erläuterung der einzelnen Zeilennummern:

10 Aufruf des Befehlseingabe-Moduls
20 Ausgabe des Befehls mit dem Variablennamen BE$
30 Zurück zum Anfang
50000 POKE 198,0 setzt den Tastaturpuffer auf Null, damit die Befehlseingabe nicht übersprungen wird. Der Befehlsstring BE$ wird gelöscht.
50020 Mit dem INPUT-Befehl wird der Befehlsstring BE$ eingelesen
50030 Ende des Unterprogramms, Rücksprung

Diese Art der Befehlseingabe hat viele Haken:

- Man kann mit dem Cursor kreuz und quer über den gesamten Bildschirm fahren.
- Bei Eingabe eines Kommas erscheint »?EXTRA IGNORED ERROR«.
- Professionelle Routinen fahren mit dem Programmablauf fort, wenn über einen längeren Zeitraum keine Eingabe erfolgt.

Sie werden zugeben, daß der INPUT-Befehl offensichtlich nicht für eine professionelle Eingaberoutine geeignet ist.

Laden Sie »LISTING 2« und starten es. Auf den ersten Blick scheint sich nichts geändert zu haben. Bei intensiverer Betrachtung (mit LIST) stellen Sie gravierende Unterschiede zum vorhergehenden Programm fest:

- Die Cursortasten sind ausgeschaltet.
- Ausschließlich folgende Tastaturzeichen werden akzeptiert: A bis Z, Anführungszeichen, <SPACE>, <DEL>, <RETURN>.

- Mit <DEL> kann nur die Eingabe gelöscht werden – nicht die Frage »WAS NUN?«.

Derartige Eingaberoutinen finden Sie in professionellen Abenteuerspielen.

Die Dokumentation zu »LISTING 2«:

50010 Tastaturpuffer und BE$ löschen. »WAS NUN?« ausgeben.
50020 Cursor einschalten.
50030 Auf Eingabe eines einzelnen Zeichens (X$) warten.
50040 Durch PEEK (203) erfährt man, welche Taste gedrückt worden ist. Wird die RETURN-Taste gedrückt oder überschreitet die Länge des Befehls BE$ 68 Zeichen, beendet das Programm die Routine: Der Cursor schaltet sich ab.
50050 Beschränkt die Möglichkeit der einzugebenden Tasten auf erwähnten Zeichen (vgl. ASCII-Werte im Commodore-Handbuch).
50060 Verhindert, daß mit <DEL> mehr Zeichen gelöscht werden als beabsichtigt.
50070 Ist die <DEL>-Taste gedrückt? Trifft dies zu, dann Cursor ausschalten. Letztes Zeichen löschen, ebenso das letzte Zeichen von BE$.
50080 Das eingegebene Zeichen X$ ausgeben. Befehlsstring BE$ um X$ erweitern. Weitere Tastatureingabe abwarten.

Eine Verbesserung stellt »LISTING 3« dar. Laden Sie es in den C64, starten und LISTen es anschließend. Erfolgt 30 s lang keine Eingabe, endet die Routine und fährt im Programm fort – die Bedenkzeit des Spielers wird eingeschränkt. Außerdem können Sie sehen, daß der Bildschirm in zwei Abschnitte unterteilt ist (Adventure mit Grafik und Text). Der obere Teil des Bildschirms dient als Grafik-Fenster, der untere Bereich zur Texteingabe.

Im Listing tauchen zwei bislang unbekannte Elemente auf:
- TI$ wird zum Registrieren der Abfragedauer eingesetzt. Sind 30 s verstrichen, fährt das Programm automatisch fort (Zeile 50020).
- Werfen Sie einen Blick auf Zeile 50000. Dort steht u.a.:

```
POKE 211,0: POKE 214,18:SYS 58732
```

Damit kann man bestimmen, wo der Cursor bzw. die nächste PRINT-Ausgabe erscheint.

POKE 214, Zeile (0 bis 24)
POKE 211, Spalte (0 bis 39)

Anschließend muß »SYS 58732« eingegeben werden. Diese Anweisung ruft eine Betriebssystem-Routine auf, die den Cursor an der neuen Stelle positioniert.

Jetzt ist Ihnen klar, wie man mit String-Funktionen umgeht und Befehlseingabe-Routinen programmiert. Experimentieren Sie mit den vorgestellten Basic-Routinen und ändern Sie sie nach Ihren Wünschen. Programmieren Sie ein eigenes Befehlseingabe-Modul mit den Zeilennummern 50000 bis maximal 50999, das Sie im weiteren Kursverlauf verwenden können.

Das Unterprogramm muß durch »GOSUB 50000« aufzurufen sein und den Befehlssatz für die Eingabe liefern (BE$).

Die Befehlszerlegung

Das nächste Programm-Modul ist der erste Schritt zur Befehlsanalyse. Worin besteht der Unterschied?

Die Befehlszerlegung zerlegt die Befehlseingabe BE$ in alle Einzelteile, die möglich sind. Das ist notwendig, da der Computer nur einen Befehl nach dem anderen bearbeiten kann. Die Befehlsanalyse führt die Einzelbefehle aus, das Programm reagiert.

Bestimmt kennen Sie das englische Adventure »The Hobbit«, die Umsetzung des erfolgreichen Romans von J. R. Tolkien für den C64. Es handelt sich um ein Adventure mit exzellenter Befehlsanalyse. Der Wortschatz kennt kaum Grenzen.

Hier ein Beispiel, ins Deutsche übersetzt:

```
SIE SEHEN: SCHWERT
SEIL
DIE FELSENTUER
MOEGLICHE RICHTUNGEN: SUEDEN
WAS NUN? NIMM DAS SCHWERT, DAS SEIL UND GEH NACH
SUEDEN.
SIE NEHMEN DAS SCHWERT.
SIE NEHMEN DAS SEIL.
SIE GEHEN NACH SUEDEN.
```

Der Computer teilt dem Spieler mit, was er sieht und welche Richtungen möglich sind. Nach der Befehlseingabe meldet sich der Computer mit einer Reaktion, als hätten Sie drei Anweisungen nacheinander eingegeben.

Unser Ziel soll sein, eine gleichartige Befehlszerlegung und -analyse zu programmieren.

Vor der Praxis kommt die Theorie. Betrachten wir folgende Befehlssätze mit je einem Befehl:

1. NIMM DAS SCHWERT.
2. GEH NACH NORDEN.
3. UNTERSUCHE DAS SCHWERT.
4. WIRF DAS SEIL.
5. OEFFNE DIE TRUHE.
6. LEGE DAS SCHWERT IN DIE TRUHE.
7. TOETE DAS MONSTER MIT DEM SCHWERT.
8. WARTE.

Dies sind typische Befehlssätze BE$. Die Eingabe wird auf das Nötigste komprimiert. Alle zusätzlichen Worte, die mit der

Sätze werden komprimiert

Befehlserkennung nichts zu tun haben, werden aussortiert. Es entstehen folgende zusammengefaßte Eingaben:

1. NIMM SCHWERT
2. N
3. UNTERSUCHE SCHWERT
4. WIRF SEIL
5. OEFFNE TRUHE
6. LEGE SCHWERT TRUHE
7. TOETE MONSTER SCHWERT
8. WARTE

Der Sinn dieser verkürzten Sätze ist eindeutig – wenn auch extrem schlechtes Deutsch. Unser zu entwickelndes Programmanalyse-Modul soll es verstehen.

Dabei fällt auf, daß die komprimierten Sätze aus maximal drei Wörtern bestehen. Machen Sie einen Test: Denken Sie sich einige komplizierte Befehlssätze aus und sortieren Sie alle überflüssigen Worte aus. Der verbleibende Rest besteht garantiert aus nicht mehr als drei Wörtern. Dies gilt nur für Befehlssätze mit einem Befehl. Es darf weder das Wort »und« noch ein Komma darin vorkommen.

Befehlssätze mit mehreren Anweisungen werden in einzelne Sätze aufgeteilt.

Aus »NIMM DAS SCHWERT UND GEH NACH NORDEN« wird »NIMM DAS SCHWERT«, »GEH NACH NORDEN«. Die beiden Sätze müssen nach dem bekannten Schema zerlegt werden. Die komprimierten Befehlssätze werden auf folgende Elemente untersucht:

Verben: Wörter wie NIMM, OEFFNE und Himmelsrichtungen (N, S usw.). Der Befehl N ist eine mögliche Abkürzung für »GEH NACH NORDEN« und wird als Verb interpretiert.

OBJEKTE: Wörter wie TUER, TRUHE, BAUM, HAUS. Es sind Dinge, die man nicht transportieren kann. Begriffe wie z.B. MONSTER, GEIST usw. ordnet man ebenfalls dieser Gruppe zu.

GEGENSTÄNDE: Objekte, die man tragen kann. (z.B. SCHWERT, SCHLUESSEL, TRUHE, SEIL)

Im weiteren Kursverlauf werde ich folgende Abkürzungen verwenden:

ve für VERB
ob für OBJEKT
ge für GEGENSTAND

Wenn wir die zweite Eingaben-Tabelle auf diese Elemente überprüfen, erhalten wir folgende Liste:
1. ve + ge
2. ve
3. ve + ge
4. ve + ge
5. ve + ob oder (bzw. ve + ge, falls die Truhe transportabel ist)
6. ve + ge + ob
7. ve + ob + ge
8. ve

Daraus läßt sich erkennen:
- jeder Befehlssatz enthält nur ein Verb
- das Verb steht an erster Stelle des Befehlssatzes
- nach dem Verb können Objekte und Gegenstände aufgeführt sein.

Üblich sind folgende »ve-ob-ge«-Kombinationen:
a) ve »WARTE«
b) ve + ge »NIMM SCHWERT«
c) ve + ob »OEFFNE TUER«
d) ve + ge + ob »LEGE SCHWERT TRUHE«
e) ve + ob + ge »TOETE MONSTER SCHWERT«
f) ve + ge + ge »VERKNOTE SEIL BRETT«

Jeder Satz beginnt mit einem Verb. Das zweite und dritte Wort kann ein Gegenstand oder ein Objekt sein. Wir haben einiges an Grammatik-Theorie durchgekaut. Sie wissen inzwischen, wie man Sätze komprimieren kann.

Kehren wir erneut zum vorhergehenden Abschnitt zurück, in dem das Befehlseingabe-Modul behandelt wurde. Dieser Programmteil muß vor dem Befehlszerlegungs-Modul aufgerufen werden.

Die anschließende Aktivierung des Befehlszerlegungs-Moduls speichert Ihre Eingabe als BE$.

```
BE$="NIMM DAS SCHWERT"
```

Der Befehlsstring BE$ wird an das Befehlszerlegungs-Modul geschickt und weiterbehandelt. Das entsprechende Programm-Schema zeigt Bild 11.

Während das Befehlseingabe-Modul lediglich als komfortable INPUT-Routine fungieren muß, hat das Befehlszerlegungs-Modul wesentlich mehr zu leisten. Drei Teilmodule erledigen folgende Aufgaben:
- String-Zerlegung
- String-Sortierung
- String-Codierung

Im Befehlszerlegungs-Modul geschieht mit dem String »BE$« folgendes:

1. Er wird in einzelne Worte aufgeschlüsselt und diesen Wörtern ein neuer String zugeordnet. Wir wollen sie mit BE$(1) bis BE$(X) bezeichnen. »X« ist die Gesamtzahl der zu ermittelnden Worte.

Für unser Beispiel gilt:

```
BE$ = "NIMM DAS SCHWERT"
BE$(1)="NIMM"
BE$(2)="DAS"
BE$(3)="SCHWERT"
```

2. Der zweite Schritt besteht darin, den Satz zu komprimieren, d.h. überflüssige Wörter werden aussortiert. Sie sind für das Verständnis des Satzsinns überflüssig, z.B. DER, DIE, DAS, DEN, DEM, UEBER, UNTER, AUF VON, IN usw.

Die Codierung unseres Eingabebeispiels sieht jetzt so aus:

```
BE$ ="NIMM DAS SCHWERT"
BE$(1)="NIMM"
BE$(2)="SCHWERT"
```

Der zweite Schritt läuft parallel zum ersten ab. Das Zerlegemodul holt sich ein Wort aus dem Befehlsstring und überprüft an Ort und Stelle, ob es überflüssig ist. Trifft dies zu, wird es nicht in die Befehlswortkette BE$(1) bis BE$(X) aufgenommen.

Für den folgenden Kursabschnitt sind Kenntnisse über den Umgang mit der DIM-Anweisung nötig.

3. Die String-Codierung wird aktiviert. Der Begriff »Code« hat in der Regel mit Zahlen zu tun. Der Trick besteht darin, den gesamten Befehlssatz in Zahlen umzuwandeln. Diese Methode spart viel Speicherplatz. Damit Sie diese Zahlen nicht durcheinander bringen, sollten Sie sich ständig Notizen machen.

Bevor der Computer Befehle analysieren kann, benötigt er einen Wortschatz, auf den er ständig zurückgreifen kann.

Ein umfangreiches Vokabular zeichnet ein gutes Adventure aus. Das Problem besteht darin, einen großen Wortschatz in möglichst wenig Speicher unterzubringen. Es gibt viele Methoden, diese Aufgabe zu lösen. Manche schlechte befindet sich darunter.

Wortgruppen-Tabellen

VERBEN			GEGENSTÄNDE		
NIMM	=	1	SCHWERT	=	1
VERLIERE	=	2	SCHLUESSEL	=	2
OEFFNE	=	3	SEIL	=	3
GIB	=	4	FACKEL	=	4
SAGE	=	5	ARMBRUST	=	5
INVENTUR	=	6	HELM	=	6
BEFESTIGE	=	7	SCHILD	=	7
OBJEKTE			**PERSONEN**		
FENSTER	=	1	GEIST	=	1
TUER	=	2	MONSTER	=	2
TRUHE	=	3	THORIN	=	3
KISTE	=	4	GOMMEL	=	4
FALLTUER	=	5	ORK	=	5

Tabelle 1. Vorgesehene Eingaben, in Gruppen unterteilt

Bestimmt kennen Sie Adventures, bei denen sich die Aufgabe des Spielers vor allem darin erschöpft, den spärlichen Wortschatz des Spiels zu erraten. Stereotyp beantworten solche Spiele die Mehrzahl der Befehlseingaben mit Antworten wie »Ich kenne dieses Wort nicht!« usw. Andere »Abenteuerspiele« bringen nicht einmal eine Fehlermeldung, sondern ignorieren den Befehl völlig. Dadurch wird der Spieler vom Adventure abgelenkt und verliert unweigerlich die Freude am Spiel.

Im Rahmen dieses Kurses möchte ich Ihnen ein Befehlsanalyse-System vorstellen, das kaum Schwächen hat und universell für alle Abenteuerspiele eingesetzt werden kann. Dabei werden im Computer Wort-Tabellen gespeichert, die allen Wörtern einer Gruppe eine Zahl zuordnen.

Wir unterscheiden folgende Wortgruppen (Tabelle 1):
- Verben
- Gegenstände
- Objekte
- Personen

Dazu brauchen wir eine zweite Tabelle, die alle auszusortierenden Wörter enthält. Davon später mehr.

Bei unserem Beispiel handelt es sich um eine kurze Tabelle. Sie kann bedeutend länger sein. Die Qualität des Adventure-Wortschatzes hängt unmittelbar damit zusammen. Wir haben vier Grundtabellen: Verben, Gegenstände, Objekte und Personen. Die Tabellen ordnen jedem darin enthaltenen

Wort eine Zahl zu. Diese Nummern wurden nicht willkürlich gewählt. Das erste Wort der Tabelle besitzt den Wert »1«, das letzte einen Wert »X« (X = Anzahl der in der Tabelle vorkommenden Worte). Auf die Programmierung der Tabelle kommen wir später.

Fest steht: Tabellen ordnen bestimmten Worten charakteristische Zahlen zu. Wir haben Ihnen empfohlen, sich vor und während der Programmerstellung laufend schriftliche Notizen zu machen. Das wird Ihnen zu diesem Zeitpunkt außerordentlich hilfreich sein.

Numerierte Wörter

Kehren wir zurück zu unserem Beispielsatz: BE$ = »NIMM DAS SCHWERT«. Das Zerlegemodul erzeugte daraus folgende Teilstrings: BE$(1)= »NIMM«, BE$(2)= »DAS«, BE$(3)= »SCHWERT«.

Nach dem Entfernen der unwichtigen Worte blieben übrig: BE$(1)= »NIMM«, BE$(2)= »SCHWERT«.

Die Zahl in Klammern hinter BE$ gibt an, wo das Wort im Satz steht - der Eingabesatz beginnt mit BE$(1). Erinnern Sie sich an unsere Grammatik-Stunde?
- Das Verb ist das erste Wort im Satz. Wir bezeichnen es mit BE$(1).
- Die sich anschließenden Worte sind Objekte, Gegenstände oder Personen.

Die Befehlscodierung arbeitet folgendermaßen: Wir nehmen das erste Wort BE$(1) und sehen in der Tabelle für die **Verben** nach, welche Ordnungszahl es besitzt. Für BE$(1) (NIMM) erhalten wir den Wert 1. Oder als Variable definiert: VERBZAHL = 1

Jetzt interessiert uns das zweite Wort. Nennen wir es BE$(2). Wir wissen nicht, ob BE$(2) ein Gegenstand, ein Objekt oder eine Person ist. Aus diesem Grund muß in allen drei Tabellen nach dem Wort gesucht werden. Als Ergebnis erhalten wir entweder eine Gegenstands-, Objekt- oder Personenzahl.

Bei dem Wort BE$(2) = »SCHWERT« ergibt sich die Gegenstandszahl »1«. BE$(2) steht in der Tabelle für die »Gegenstände« an erster Stelle.

Die gesamte Codierung der Eingabe BE$ = »NIMM DAS SCHWERT« hat folgende Werte geliefert:
VERBZAHL = 1
GEGENSTANDSZAHL = 1
OBJEKTZAHL = 0
PERSONENZAHL = 0

Damit haben Sie eine Möglichkeit kennengelernt, wie Sätze anhand von Zahlen ausgedrückt werden können. Dazu ist erforderlich, daß diese innerhalb von Worttabellen definiert sein müssen.

Unser Beispiel hat als Wert für die Objekt- und die Personenzahl jeweils »0« ergeben. Das rührt daher, daß in unserem Ausgangssatz »NIMM DAS SCHWERT« weder eine Person noch ein Objekt vorkommt. Das Schwert ist ein **Gegenstand**.

Objekte sind Dinge, die nicht transportiert werden können.

Dieses Analyse- bzw. Codierungssystem hat einen Haken: Was passiert, wenn in einem Satz zwei Gegenstände auftreten? Ein Beispiel:

```
BE$= "VERKNOTE DAS SEIL AM SCHWERT".
```

In diesem Satz kommen zwei Gegenstände vor (Seil und Schwert). In der Tabelle für die Gegenstände besitzt »SCHWERT« den Wert »1« und »SEIL« den Wert »3«. Welche der beiden Codezahlen gilt? Um dieses Problem zu lösen, müssen wir auf die Tabelle zurückkommen, die alle möglichen »ve-ob-ge«-Kombinationen darstellt.

Diese Liste sagt folgendes aus:
- Ein Befehlssatz enthält maximal ein Verb (1 Verbzahl).
- Ein Befehlssatz enthält maximal ein Objekt (1 Objektzahl).
- Ein Befehlssatz kann bis zu maximal zwei Gegenstände enthalten: Wir brauchen demnach zwei Gegenstandszahlen. Wo bleiben die Personen?

Sie haben sicher bemerkt, daß ich zur Demonstration von Schulbeispielen bewußt auf das klassische Adventure »The Hobbit« zurückgreife. Darin treten extrem viele Personen auf. Drei Eingabebeispiele:
- SAGE THORIN »GIB MIR DIE KARTE«.
- SAGE ELROND »LIES DIE KARTE«.
- SAGE BARD »ERSCHIESSE DEN DRACHEN«.

Der erste und zweite Satz sind eindeutig. Es kommt nur eine Person darin vor, zur Codierung wird lediglich **eine** Personenzahl benötigt. Im dritten Beispiel finden Sie zwei Personen - Bard und den Drachen. Sie werden staunen: Es wird wiederum nur **eine** Personenzahl benötigt.

Der Grund ist, daß im Befehlssatz lediglich eine Person betroffen ist - Bard. Der Drache erscheint in der Anweisung, die für Bard gilt.

Damit lernen Sie ein Beispiel eines indirekten Befehls kennen. Gaben wir bisher dem Computer direkte Befehle, erhält er nunmehr die Anweisung, einer bestimmten Person etwas zu »sagen«. Der Befehl für die Person Bard »ERSCHIESSE DEN DRACHEN« wird im Programm nicht von dem Modul behandelt, das wir gerade aufbauen, sondern an ein Unter-

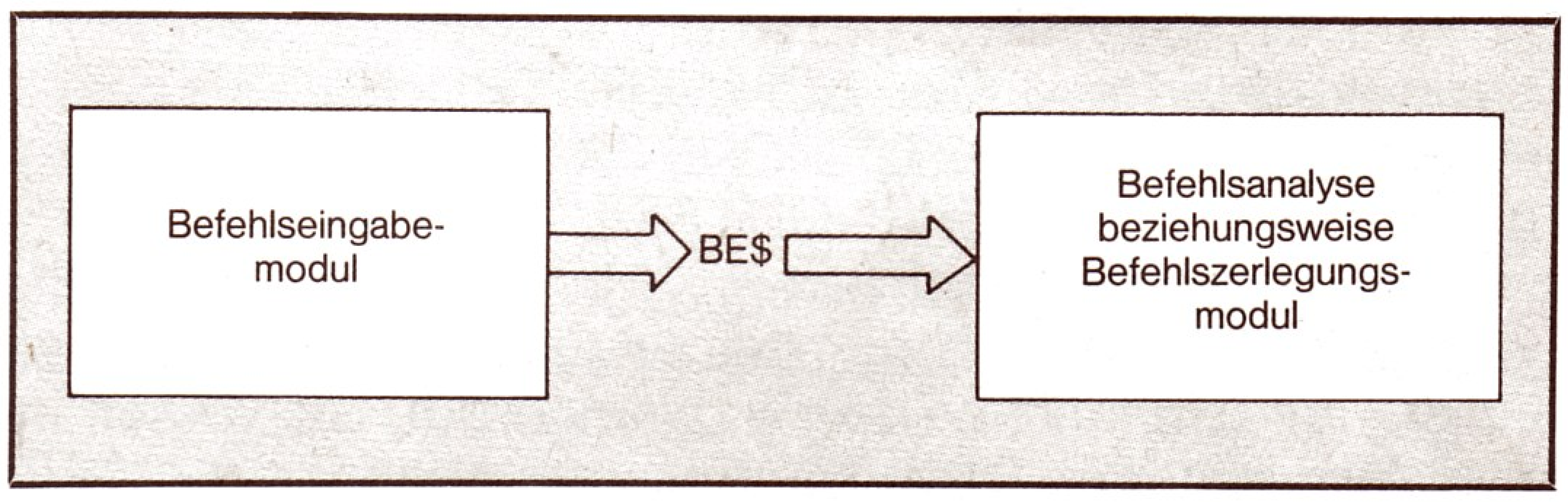


Bild 11. Zwei wichtige Programm-Module: Befehlseingabe und Befehlszerlegung

modul weitergegeben. Es ist für die Reaktionen der Personen verantwortlich. Davon später mehr.

Sie haben gelernt, daß im Modul für die direkten Befehle eine Personenzahl benötigt wird. Es kann nur jeweils eine Person angesprochen werden.

Lassen Sie uns die Werte zusammenfassen, die bei der Codierung eines Befehlssatzes ausgegeben werden:
VERBZAHL
1. GEGENSTANDSZAHL
2. GEGENSTANDSZAHL
OBJEKTZAHL
PERSONENZAHL

Dies sind die ersten Schritte auf dem Weg zur kompletten Befehlsanalyse. Bild 12 zeigt das Ablaufschema unseres neuen Moduls (Befehlszerlegungs- und Codierungs-Modul) in seinen Grundzügen. Verschiedene Übungsbeispiele zu diesem Thema sollen den Ablauf verdeutlichen.

Nach der Eingabe und einer Aktivierung des Unterprogramms »Codiermodul« erhalten wir gemäß Wortgruppen-Tabelle (Tabelle 1) als Ausgabe:

1. Befehlseingabe:

```
BE$= "OEFFNE DIE TUER"
```

VERBZAHL = 3
1. GEGENSTANDSZAHL = 0
2. GEGENSTANDSZAHL = 0
OBJEKTZAHL = 2
PERSONENZAHL = 0

2. Befehlseingabe:

```
BE$="VERLIERE DIE FACKEL"
```

VERBZAHL = 2
1. GEGENSTANDSZAHL = 4
2. GEGENSTANDSZAHL = 0

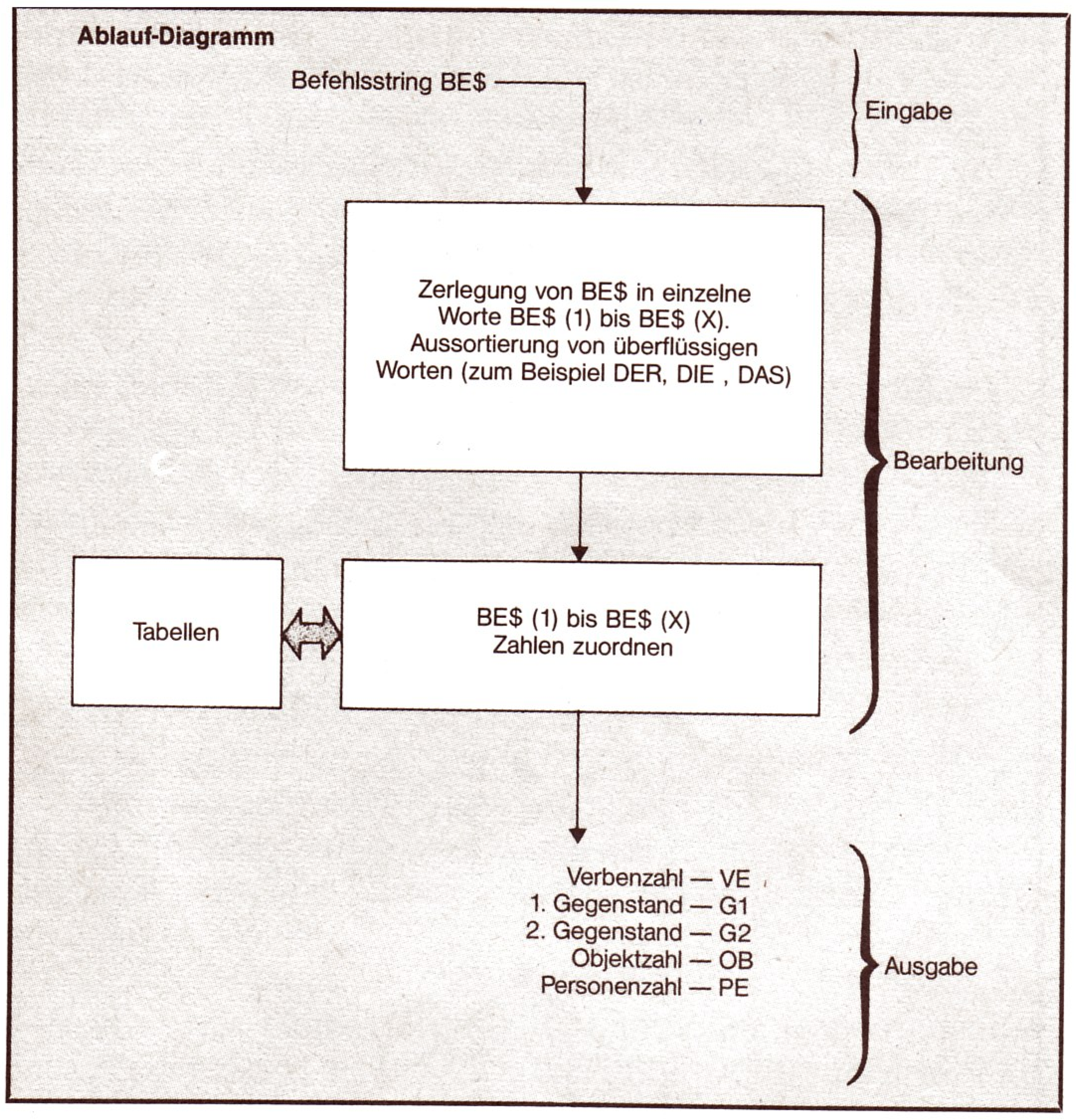


Bild 12. Funktionsweise des Codierungsmoduls

OBJEKTZAHL = 0
PERSONENZAHL = 0

3. Befehlseingabe:

```
BE$="GIB DEM GEIST DEN SCHLUESSEL"
```

VERBZAHL = 4
1. GEGENSTANDSZAHL = 2
2. GEGENSTANDSZAHL = 0
OBJEKTZAHL = 0
PERSONENZAHL = 1

4. Befehlseingabe:

```
BE$="BEFESTIGE DAS SEIL AM FENSTER"
```

VERBZAHL = 7
1. GEGENSTANDSZAHL = 3
2. GEGENSTANDSZAHL = 0
OBJEKTZAHL = 1
PERSONENZAHL = 0

Soviel zur Theorie des neuen Moduls. Später werden wir Fallbeispiele konstruieren, die aus langen Sätzen bestehen. Außerdem werden sie das Bindewort »UND« enthalten.

Das Zerlege-Chiffrier-Modul

Bevor das Modul arbeiten kann, müssen die Tabellen programmiert werden.

Verbtabelle

Die einzelnen Verben werden in DATA-Zeilen abgelegt:

```
DATA NIMM, VERLIERE, OEFFNE, GIB, SAGE, INVENTUR,
BEFESTIGE
```

Für die Anzahl der Verben muß eine numerische Variable definiert werden. Sind Sie mit »VZ« einverstanden? Diese Variablenbezeichnung darf nicht mit der Ordnungszahl für die Verben »VE« laut Tabelle 1 verwechselt werden.

Addieren wir die Anzahl der Verben in unserer Beispieltabelle: VZ = 7.

Dazu müssen wir ein Variablenfeld in Form eines String indizieren: VE$(VZ).

Die Basic-Anweisung für das Dimensionieren von Variablenfeldern lautet für unser Beispiel:

```
DIM VE$(VZ)
```

Das Feld muß mit Daten gefüllt werden:

```
FOR I=1 TO VZ : READ VE$(I) : NEXT I
```

Nach diesem Verfahren werden alle Tabellen programmiert.

Für die Verb-Tabelle sieht das wie folgt aus:

```
VE$(1) = "NIMM"
VE$(2) = "VERLIERE"
VE$(3) = "OEFFNE"
VE$(4) = "GIB"
VE$(5) = "SAGE"
VE$(6) = "INVENTUR"
VE$(7) = "BEFESTIGE"
```

Angenommen, wir haben als Eingabe-Verb

```
BE$(1)="OEFFNE"
```

Wenn Sie wissen möchten, welche Verbzahl BE$(1) hat, müssen Sie so verfahren:

```
10 FOR I=1 TO VZ
20 IF BE$(1) = VE$(I) THEN VE=I
30 NEXT I
40 PRINT"VERBZAHL = ";VE
```

Wir ändern Zeile 20, damit Abkürzungen akzeptiert werden:

```
20 IF BE$(1) = LEFT$(VE$(I),LEN(BE$(1)) THEN VE=I
```

Der Spieler gibt statt des vollständigen Wortes »VERLIERE« nur »V« ein. Das Programm versteht trotzdem, was gemeint ist, sofern die Abkürzungen eindeutig sind. Will man das Verb »SUCHE« mit »S« abkürzen, darf in der Verbtabelle kein anderes Wort stehen, das ebenfalls mit »S« beginnt.

Ist dies der Fall, greift das Modul auf das Verb zu, das es in der Tabelle als erstes mit dieser Voraussetzung findet. Verben müssen zur richtigen Interpretation unverwechselbar abgekürzt werden.

Laden Sie »LISTING 4«. Wenn Sie das Programm mit RUN starten, wird die Eingabe eines Befehls BE$ verlangt. Drücken Sie anschließend <RETURN>. Der Bildschirm gibt aus, wie der Befehlsstring BE$ in die indizierten Strings BE$(1) bis BE$(10) zerlegt wird. Das Programm »LISTING 4« ist für Eingabefolgen von maximal zehn Wörtern vorgesehen. Geben Sie verschiedene Sätze ein und beobachten, was geschieht. Dieses »Zerlegungs«-Modul hat einen Haken: Lassen Sie zwischen den Worten mehrere Leerzeichen (SPACE), interpretiert sie das Programm als eigenständige Worte. Diesen Fehler werden wir später abstellen.

Das Modul sollte im Programm ab Zeile 51000 stehen (hinter dem Unterprogramm für die Befehlseingabe).

Dokumentation der Programmzeilen zu »LISTING 4«:

10 – 70 Zur Demonstration des Moduls ab 51000.
51000 Beginn des Zerlege-Moduls.
51010 Das Feld für die einzelnen Wörter des Befehlssatzes wird gelöscht.
51020 Die Wort-Zählvariable WZ beginnt bei »1«.
51030 Diese Schleife durchläuft den Befehlsstring BE$ vom ersten bis zum letzten Zeichen.
51040 Auf ein Leerzeichen (SPACE) prüfen. Trifft dies zu, interpretiert es das Modul als Beginn eines neuen Wortes. Die Wort-Zählvariable WS erhöht sich um »1«.
51045 Verhindert ein Überschreiten der zulässigen Anzahl der Worte (10).
51050 Der aktuelle Eingabestring BE$(WZ) wird aufgebaut.
51060 Sprung zum Schleifenanfang (Zeile 51030).
51070 Ende des Unterprogramm-Moduls, Rücksprung.

Wir erhalten die Werte BE$(1) bis BE$(WZ) (die einzelnen Wörter von BE$) und WZ (den Variablenwert für die Anzahl der Wörter).

Die Funktionsweise des Moduls ist leicht verständlich, wenn Sie sich das Listing auf dem Bildschirm ansehen.

Der nächste Schritt ist das Aussortieren überflüssiger Wörter. Sie werden nicht in die Wortkette BE$(1) bis BE$(10) aufgenommen.

Eine verbesserte Version finden Sie als »LISTING 5« auf der beiliegenden Diskette. Laden Sie das Programm.
Hier die Dokumentation der Programmzeilen:

5	Das Unterprogramm für die Tabellen-Definition wird aufgerufen.
10 – 70	Zur Programm-Demonstration.
51000 – 51070	Wie »LISTING 4«, mit folgender Änderung:
51040	Suche nach einem Leerzeichen (SPACE). Der Unterschied zu »LISTING 4«: Anstatt das Leerzeichen als neues Wort zu interpretieren, wird ein Unterprogramm ab Zeile 51100 aufgerufen. Es dient zur Aussortierung unnötiger Worte und entscheidet, ob das aktuelle Wort BE$(WZ) in die Wortkette BE$(1) bis BE$(10) übernommen wird.
60000 – 60015	DATA-Zeilen und Einlese-Schleife der auszusortierenden Wörter. Das Stringfeld bezeichen wir mit AU$, die Anzahl der Worte mit der Variablen AZ.

Die Erläuterung des Unterprogramms:

51100	REM-Zeile, Start
51110	Die Check-Variable IC wird auf »0« gesetzt. Schleife »I1« liest die AU$-Tabelle.
51120	Kommt das aktuelle Wort BE$(AW) in der Tabelle AU$ vor? Wenn ja, aussortieren. Check-Variable IC auf »1« setzen.
51130	Sprung zum Schleifenbeginn
51140	Ist der Schleifendurchlauf beendet, wird der Wert von IC überprüft. Steht er bei »0«, kommt das aktuelle Wort nicht in der AU$-Tabelle vor. Das Aussortieren erübrigt sich. Es wird in die Wortkette BE$(1) bis BE$(10) übernommen, der Wortzähler WZ um »1« erhöht und das Unterprogramm beendet.
51150	Die Bedingung in Zeile 51140 wurde nicht erfüllt. Demnach muß es sich bei dem aktuellen Wort BE$(AW) um ein Wort der AU$-Tabelle handeln und eliminiert werden. BE$(AW) wird gelöscht, der aktuelle Wortzähler WZ nicht erhöht. Die Check-Variable IC dient zum Überprüfen von Bedingungen.

Starten Sie das Programm mit RUN. Geben Sie mehrere Befehlssätze ein. Untersuchen Sie die Ergebnisse. Falls die Eingabe absichtlich aus wirren Buchstabenketten besteht, die nicht in der AU$-Tabelle vertreten sind, werden Sie trotzdem akzeptiert. Sehen Sie sich die DATA-Zeile 60005 (AU$-Tabelle) an. Deutlich erkennbar finden Sie dort auch ein Leerzeichen. Dies ist nötig, um eine unsaubere Programmierung (mehrere SPACEs zwischen den Wörtern in BE$) zu vermeiden: Leerzeichen werden nicht mehr als Worte akzeptiert.

Zur Übung empfehle ich, die AU$-Tabelle zu erweitern. Tippen Sie ab Zeile 60005 neue DATAs dazu. Ändern Sie die Variable AZ in den gültigen Wert.

Die Wort-Codierung

Bisher haben wir Befehlssatz-Beispiele behandelt, die nur einen Befehl enthielten (ohne »UND«). Unser Programm sollte aber die Eingabe mehrerer Befehle verarbeiten können.

Kehren wir zum Eingabebeispiel aus »The Hobbit« zurück. Deutlich ist zu erkennen, daß beim Analyse-System ein Befehl nach dem anderen ausgeführt werden kann. Am besten erkennt man das daran, daß der Computer auf einen Befehlssatz wie »Nimm das Schwert und das Seil« nicht mit »Sie nehmen das Schwert und das Seil« antwortet, sondern:

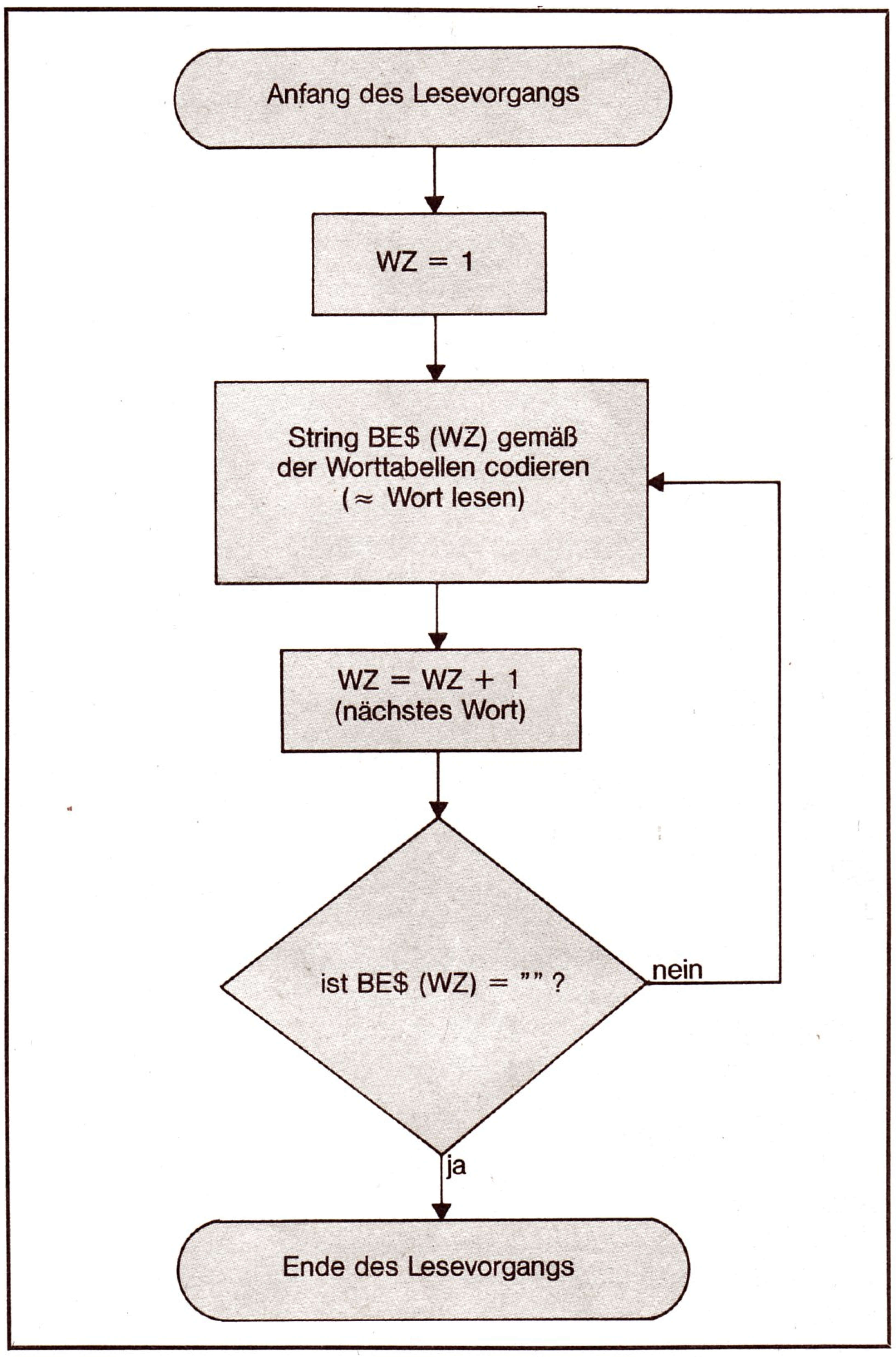


Bild 13. Programmablauf beim Codieren eines Satzes mit zwei Befehlen

»Sie nehmen das Schwert«
»Sie nehmen das Seil«

Untersuchen wir folgendes Eingabebeispiel für mehrere Anweisungen:

```
BE$= "NIMM DAS SCHWERT UND DAS SEIL UND GEH
NACH NORDEN"
```

Nachdem das Modul den Satz zerlegt und überflüssige Wörter AU$ aussortiert hat, liegt uns dieses Ergebnis vor:

```
BE$(1) = "NIMM"
BE$(2) = "SCHWERT"
BE$(3) = "UND"
BE$(4) = "SEIL"
BE$(5) = "UND"
BE$(6) = "NORDEN
```

Zwei neue Wörter fallen uns auf: UND, NORDEN. »UND« soll nicht aussortiert werden. Wundern Sie sich, weil ich das Wort »NORDEN« nicht berücksichtigt habe? Richtungsangaben sind Ausnahmefälle. Davon später mehr.

Um die Angelegenheit nicht unnötig zu komplizieren, wollen wir uns um die Variablen BE$(5) und BE$(6) im Augenblick nicht kümmern. Es bleibt BE$(1) bis BE$(4) als Eingabesatz mit zwei Anweisungen.

Das Programm liest sequentiell ein Eingabewort nach dem anderen. Wir benötigen dazu die Wortzähl-Variable WZ. Vor Beginn des Lesevorgangs wird sie auf »1« gesetzt. Der Computer erhält das Wort BE$(WZ).

Nach dem Codieren wird der Wortzähler um »1« erhöht (WZ=WZ+1). Es beginnt die Überprüfung, ob BE$(WZ) existiert. Enthält er Buchstaben, Zahlen oder ist es ein Leerzei-

chen? Wird er als solches identifiziert, setzt das Programm den Lesevorgang fort. Andernfalls wird dieser beendet. Das Ablaufschema der Leseroutine finden Sie in Bild 13. Achtung: Mit LESEN meine ich in dem Fall CODIEREN!

Das Modul ist unvollständig. Es versteht lediglich Zwei-Wort-Befehle. Um dies zu verdeutlichen, müssen wir einen Blick auf das gesamte Ablaufschema des Adventures werfen (Bild 14).

Der Bereich des Befehlseingabe- und Codiermoduls (mit Zerlegung und Aussortierung) ist mit unterbrochenen Linien umrandet. Wir planen, aus den beiden Modulen ein zusammenhängendes Unterprogramm zu schaffen, das für jedes Adventure universell eingesetzt werden kann. Der gestrichelte Bereich zeigt exakt das Ablaufschema des Lesevorganges, den wir zuvor besprochen haben.

Erläuterung des Programmablauf-Plans

Der Ablaufplan ist leicht verständlich dargestellt. Zu Beginn des Abenteuerspiels erscheint das Titelbild und die Spielanleitung. Tabellen werden eingelesen, eventuell Sprites definiert usw. Das Spiel beginnt. Befehlseingabe- und Codiermodul werden aktiviert.

Das Action-Modul nimmt seine Arbeit auf. Es bildet das Kernstück, die »Intelligenz« des Adventures, und unterscheidet sich von Spiel zu Spiel. In diesem Programmteil wird auf Befehle reagiert, Lagebericht erstattet, werden Personen (Monster, Geister, Wachen usw.) gesteuert, bewegt, zum Leben erweckt. Das Action-Modul stellt die Aufgaben für den

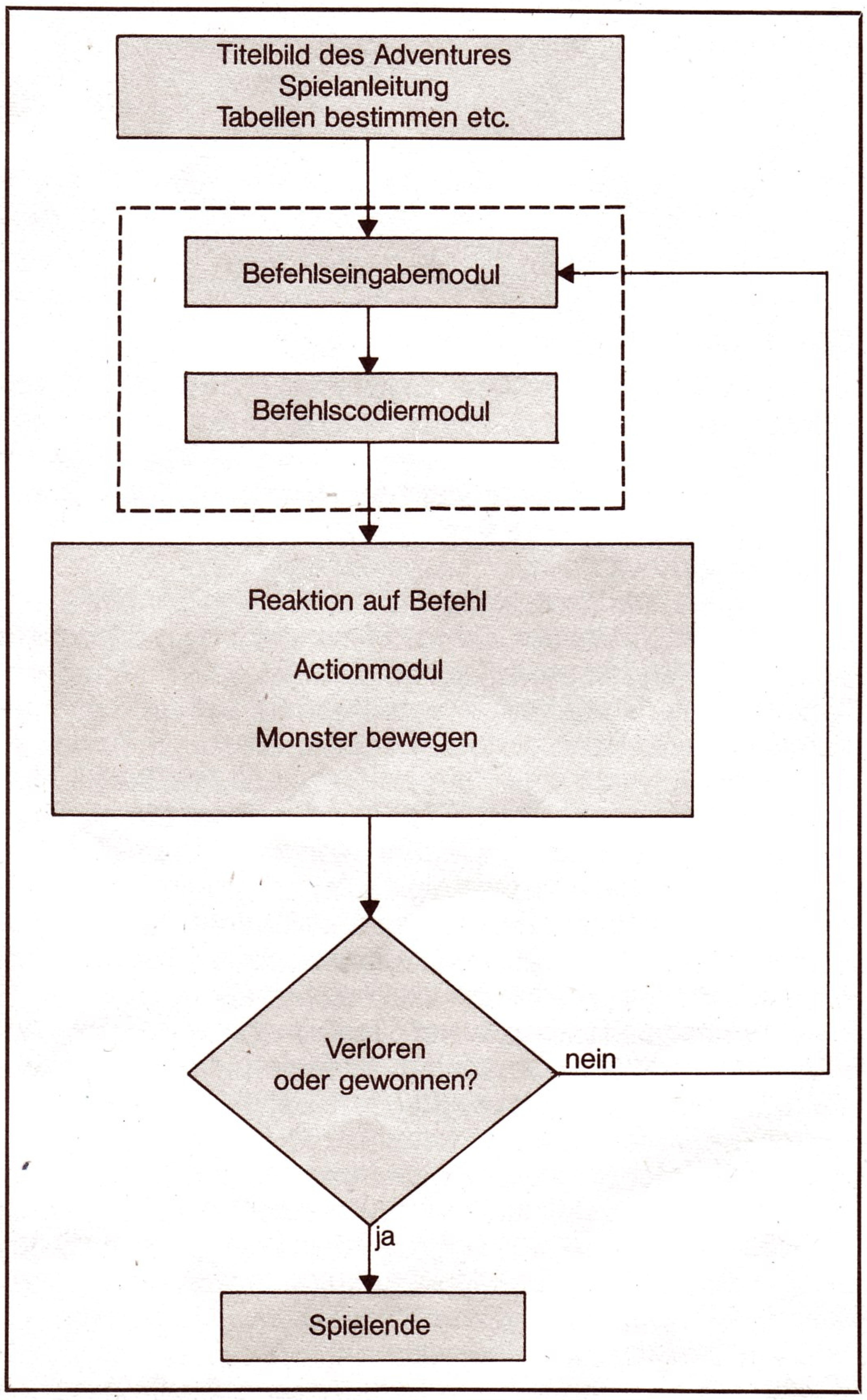


Bild 14. Der Ablaufplan unseres Adventures

Spieler und verwaltet die Spielkarte. Es behält genauen Überblick, wo welcher Gegenstand liegt, wo sich jede Person befindet, was sie gerade macht, welche Türen offen oder geschlossen sind. Der Umfang dieses Moduls bestimmt letztendlich Qualität und Vielfalt des Spiels. Die abschließende Aufgabe des Action-Moduls ist eine Überprüfung, ob das Spiel zu Ende ist (verloren oder gewonnen) oder ob es fortgesetzt werden soll (zurück zum Befehlseingabe- und Codiermodul).

Das einfache Grundprinzip eines Adventures wird Ihnen nach dem bisherigen Kursverlauf vertraut sein. Ich möchte Sie anregen, sich eigene Abenteuerspiele auszudenken und sie problemlos und ohne großen Aufwand in ein Programm umzusetzen.

Das Action-Modul

Nachdem das Befehlszerlegungs-Modul den Eingabestring sortiert und in einzelne Wörter zerpflückt hat, gibt es BE$(1) bis BE$(X) an das Codiermodul weiter, wie z.B. folgende vier Wörter:

```
NIMM SCHWERT UND SEIL
```

Das Codiermodul liest diesen Satz Wort für Wort. Welches momentan bearbeitet wird, bestimmt die Variable WZ (Wortzähler). Damit der Lesevorgang beim ersten Wort beginnt, setzten wir die Wortzähl-Variable WZ am Anfang des Lese- bzw. Codiermoduls auf »1«. BE$(WZ) wird auf folgende Weise codiert:

Die Check-Variable IC und die Wortvariablen beginnen mit »0«.

```
IC = 0 : VE=0 : G1=0 : G2=0 : OB=0 : PE=0
```

Es wird überprüft, ob der String BE$(WZ) in der Verb-Tabelle enthalten ist:

```
FORI: 1 TO VZ : REM VZ = ANZAHL DER VERBEN
IF BE$(WZ) = VE$(I) THEN IC=1 : VE=I
NEXT I
```

Findet das Modul ein Verb, wird es mit der Verbzahl VE codiert und die Check-Variable IC auf »1« gesetzt. Auf diese Art werden alle anderen Wort-Tabellen durchsucht (Gegenstände, Objekte, Personen).

Sie könnten einwenden, daß wir für das erste Wort in der Verb-Tabelle nachsehen müssen. Es gilt die Regel, daß das erste Wort eines Satzes nur ein Verb sein kann. Damit haben Sie recht. Trotzdem schlage ich die genannte Methode vor. Um die Arbeitszeit des Moduls zu verkürzen, kann man nach jedem Tabellendurchlauf überprüfen lassen, ob sich die Check-Variable auf »1« geändert hat. Trifft diese Bedingung zu (IC=1), werden die restlichen Tabellen übersprungen. Ein Wort, das in Tabelle A gefunden wurde, ist niemals in Tabelle B vertreten. Es gibt kein Wort, das gleichzeitig Verb und Person ist. Sind alle Tabellen durchlaufen, wird die Check-Variable IC erneut auf ihren Inhalt überprüft. Ist IC = 0, kommt das Wort BE$(WZ) in keiner Tabelle vor. Eine Fehlermeldung erscheint, z.B. »Mein Wortschatz kennt das Wort XYZ nicht!«. Die Befehlscodierung bricht ab. Das Programm hat jetzt folgende Möglichkeiten:

1. Es springt zurück zum Action-Modul. Alle Monster, Geister usw. sind wieder am Zug. Stellen Sie sich folgende Spielsituation vor: Der Spieler begegnet einem Monster und gibt als Befehl »Ermorde das Monster« ein. Das Programm kennt »ermorde« nicht. Das Monster frißt Sie – das Abenteuerspiel ist für diese Runde beendet.
2. Das Programm springt zurück zum Befehlseingabemodul.

Methode 1 ist die gebräuchlichste. Ich halte es für sinnvoller, Methode 2 zu verwenden. Damit ist sichergestellt, daß der Spieler nicht aufgrund eines Mißverständnisses verliert. Ein gutes Adventure läßt den Spieler durch taktische Fehlentscheidungen verlieren, nie aufgrund von Tippfehlern.

Nehmen wir an, es tritt keine Fehlermeldung auf und das Programm fährt in seinem Ablauf fort. Die Check-Variable IC ist »1«. Damit wurde eine Wortzahl (Verbzahl VE, 1. Gegen-

standszahl G1, usw.) gefunden. Beim ersten Wort (WZ = 1) handelt es sich normalerweise um die Verbzahl VE. Wir haben das erste Wort codiert. Die Wortzählvariable erhöht sich um »1«. Das nächste Wort wird bearbeitet. Das Programm überprüft, ob es als BE$(WZ) anerkannt werden kann (enthält es Zeichen?). Wenn nicht, ist der Lesevorgang beendet und das Action-Modul ist an der Reihe. Gehen wir davon aus, daß BE$(WZ) als interpretierbares Wort existiert. Nach unserem Ablaufschema wird der Lesevorgang wiederholt.

Was passiert, wenn es sich bei dem neuen String BE$(WZ) um das Bindewort »UND« handelt? Das ist ein neuer Befehl innerhalb des Befehlssatzes. Nach »UND« folgt in der Regel ein neues Verb oder ein Objekt. Ist das nächste Wort ein Verb (z.B. NIMM SCHWERT UND **OEFFNE** TUER), würde sich unsere Verbzahl ändern und die NIMM-Anweisung verlorengehen. Die Lösung des Problems ist nicht schwer.

Vor dem erneuten Lesevorgang überprüft das Programm, ob es sich um das Wort »UND« handelt. Es muß in keiner Tabelle definiert werden. Wir stellen die direkte Frage:

```
IF BE$(WZ) = "UND"...
```

Trifft dies zu, setzen wir die Variable UD auf »1«.

Der Lesevorgang endet. Es folgt der Sprung zum Action-Modul, das auf den ersten Befehl des Eingabesatzes reagiert. In unserem Beispiel meldet sich der Computer mit:

```
SIE NEHMEN DAS SCHWERT
```

Dann kehrt das Programm zum Befehlseingabemodul zurück.

Stop! Wir können nicht einfach einen neuen Befehlsstring BE$ vom Spieler holen, wenn der alte BE$ noch nicht vollständig ausgewertet ist. Deshalb muß in der ersten Zeile des Befehlseingabemoduls abgefragt werden, welchen Zustand die Variable UD hat. Hat sie den Wert »1«, ist der letzte Befehlsstring unvollständig bearbeitet. Das Befehlseingabemodul wird übersprungen. Das Programm soll zum Lese-(Codier-)Modul verzweigen. Dabei darf der Wortzähler WZ nicht wie zuvor auf »0« gesetzt werden. Das Modul soll an der Stelle im Satz weiterlesen, an der unterbrochen wurde (nicht erneut ab erstem Wort des Satzes).

Auch Verbzahl VE, Objektzahl OB und Personenzahl PE dürfen nicht gelöscht werden. Würde man bei unserem Eingabebeispiel »NIMM SCHWERT UND SEIL« nach der Befehlsausführung von NIMM SCHWERT die Verbzahl löschen, wüßte der Computer nach Beendigung des zweiten Lesevorgangs lediglich, daß ein Objekt (SEIL) gemeint ist. Was er damit anfangen soll, würde ihm für immer ein Rätsel bleiben. Wird ein weiteres Verb gefunden, erhält die Verbzahl den

Intelligentes Modul

Wert des neuen Verbs und »vergißt« die bisher gültige Zahl. Das Verfahren gewährleistet die korrekte Interpretation aller UND-Zusammenhänge. Diesen bemerkenswerten Vorteil unserer Befehlscodierung bieten wenige Adventures.

Die ersten Zeilen des Codiermoduls müssen lauten:

1. Zeile: IF UD=1 THEN UD=0 : GOTO 3.Zeile
2. Zeile: WZ=1 : VE=0 : OB=0 : PE=0
3. Zeile: IC=0 : G1=0 : G2=0

In der ersten Zeile wird UD erneut auf »0« gesetzt, um das Befehlseingabemodul nicht zu überspringen. Findet das Programm beim zweiten Lesevorgang ein weiteres »UND«, wird die dafür vorgesehene Variable UD wieder auf »1« gesetzt.

Das verbesserte Ablaufschema läßt sich mit Hilfe von Bild 15 nachvollziehen. Unser Grundschema für Adventures ändert sich geringfügig. Bild 16 zeigt Ihnen den neuen Ablaufplan des gesamten Programms.

Unser neues Modul (es vereint Befehlseingabe, Zerlegung, Sortierung und Codierung), steht im Programm an gewohnter Stelle ab Zeile 50000. Dieser Programmteil präsentiert sich

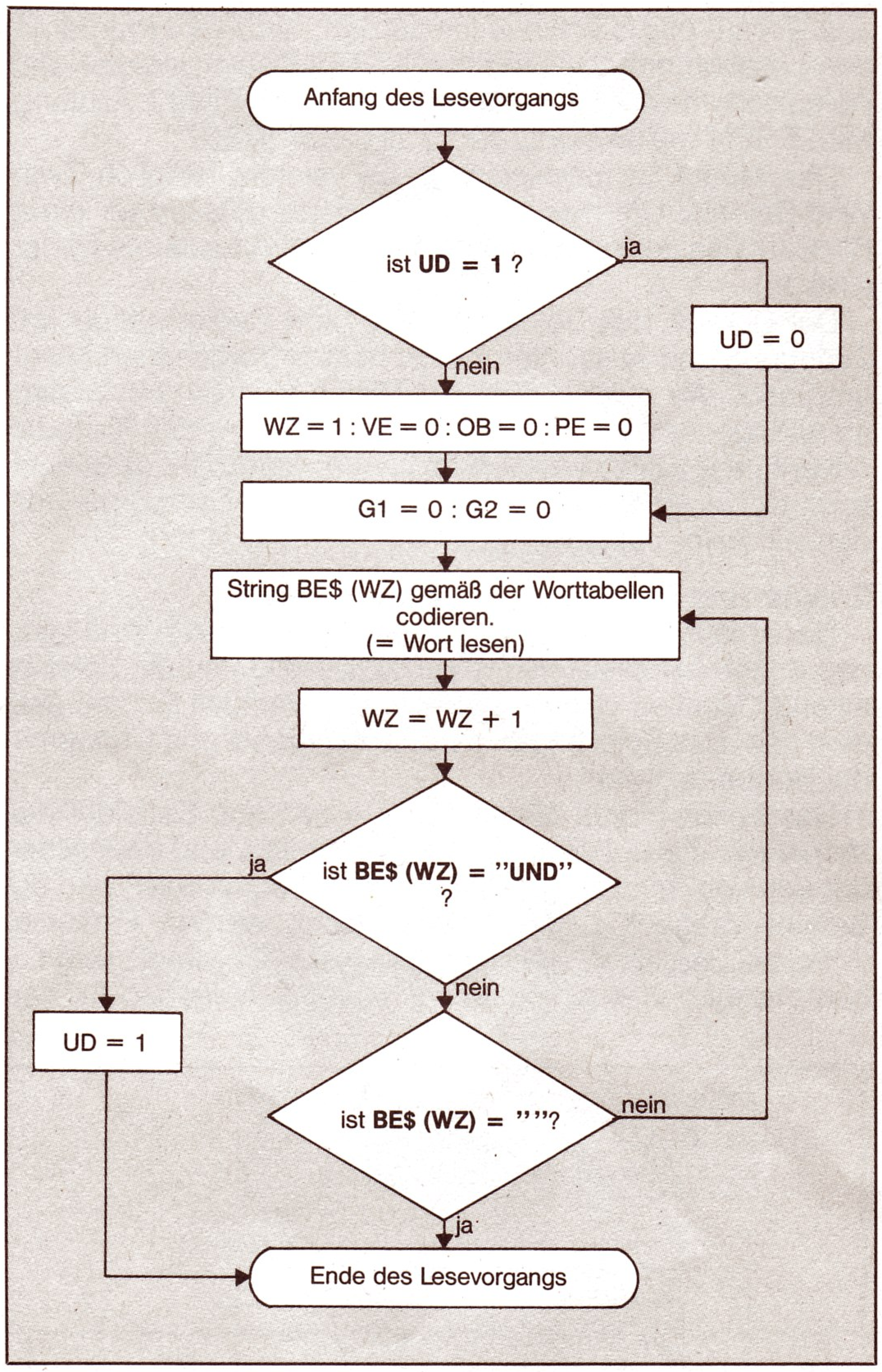


Bild 15. Das verbesserte Codierungsmodul

bereits in »intelligenter« Form: Er kann mit GOSUB 50000 aufgerufen werden und selbständig über seine Vorgehensweise beim Interpretieren einer Befehlseingabe entscheiden. Diese Intelligenz befähigt das Modul zum Einsatz im jedem Abenteuerspiel. Lediglich der Wortschatz (die Worttabellen) müssen geändert werden.

Laden Sie das Programm »LISTING 6« von der beiliegenden Diskette und lassen sich das Listing auf dem Bildschirm ausgeben. Alle Variablen, die darin auftauchen, finden Sie in Tabelle 2. Die Variablennamen werden wir während des gesamten Kursverlaufes beibehalten. Auf eventuelle Änderungen weise ich gesondert hin und ergänze die Tabelle. Ich empfehle Ihnen, bei Ihren eigenen Adventures die gleichen Variablen zu verwenden. Dadurch versetzen Sie sich in die Lage, problemlos Abenteuerspiele zu programmieren. Man kann sich eine Karte anlegen, in der alle Variablen eines Adventures verzeichnet sind. Diese Tabelle sollten Sie beim Programmieren als Hilfe neben sich liegen haben.

Ich würde mich freuen, wenn alle »Teilnehmer« unseres Adventure-Kurses die genannten Variablennamen übernehmen. Es läßt sich damit ein Standard für die Adventure-Programmierung festlegen.

Die Dokumentation der Programmzeilen zu »LISTING 6«:

5	Definition der Wort-Tabellen.
10 – 80	Steuerung des Adventures. Hier muß das Action-Modul aufgerufen werden.
50000	Anfang des Moduls.
50000 – 50030	Befehlseingabe.
50010	Ist UD=1 (der letzte Eingabestring enthält mehrere Befehle, die noch nicht ausgeführt

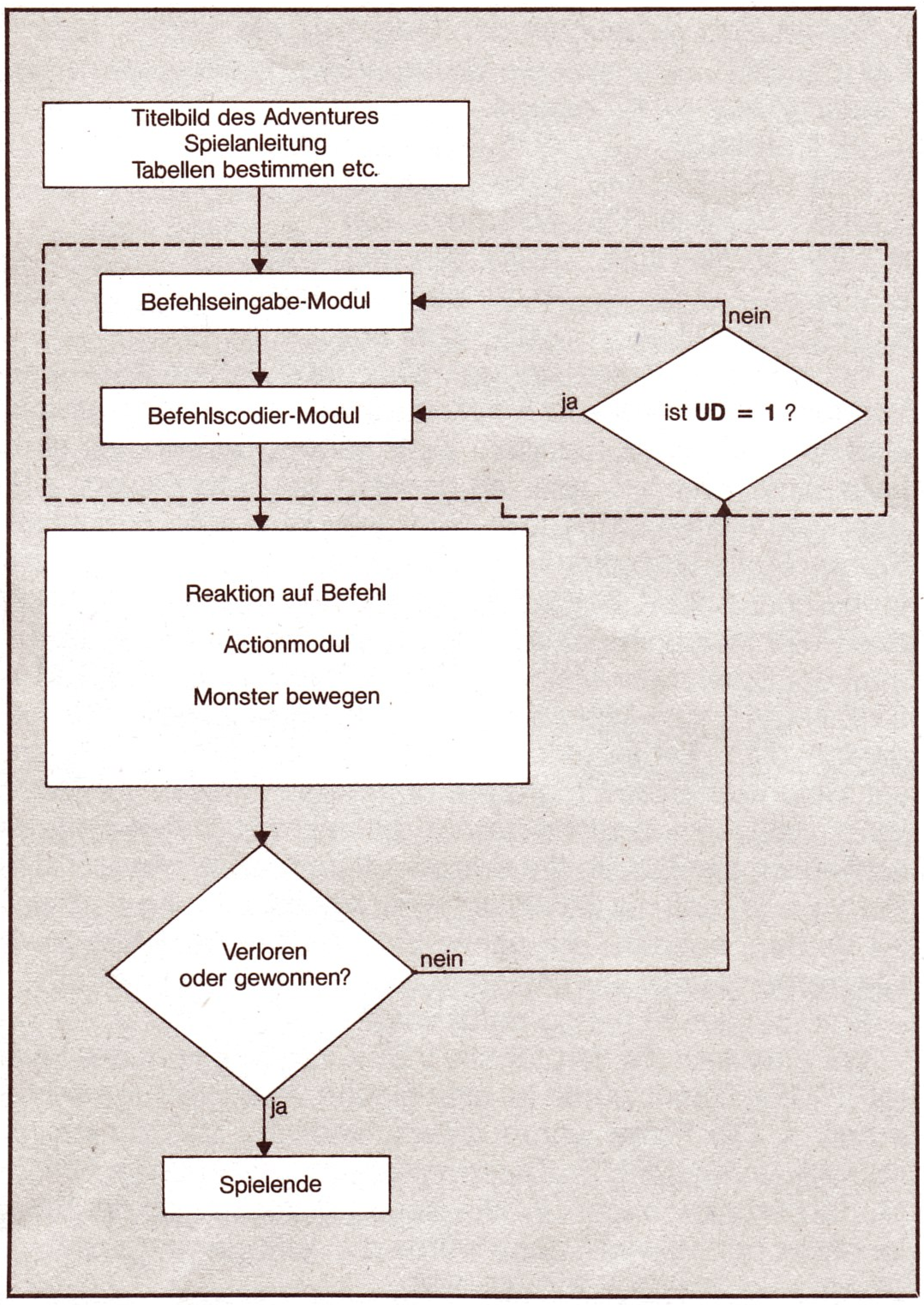


Bild 16. Geändertes Grundablaufschema des Adventures

wurden), wird die Befehlseingabe übersprungen und der alte Befehlsstring bearbeitet.

51000 – 51150 Zerlegung des Befehlssatzes BE$ in einzelne Worte BE$(1) bis BE$(maximal 10).

52000 – 52170 Befehlscodierung (Lesevorgang).

52005 Ist UD=1, handelt es sich um die Weiterbearbeitung eines alten Befehlssatzes. Der Wortzähler darf nicht auf »1« gesetzt werden. Das Programm liest dort weiter, wo zuvor abgebrochen wurde. Damit Sätze wie »NIMM DAS SCHWERT UND DAS SEIL« verstanden werden, dürfen die Variablen VE, OB, PE nicht gelöscht werden.

52025 – 52135 Hier werden alle Worttabellen auf das aktuelle Wort BE$(WZ) untersucht. Wird es in der Verb-Tabelle gefunden, erhält die Verbzahl VE ihren Wert usw. Findet man es in einer anderen Tabelle, wird die Checkvariable IC auf »1« gesetzt. Nach jedem Tabellendurchlauf wird IC überprüft. Ist IC=1, werden alle weiteren Tabellendurchläufe übersprungen, da ein Wort niemals in zwei Tabellen gleichzeitig vertreten ist. Das Überspringen dient dazu, die Verarbeitungsgeschwindigkeit des Moduls zu erhöhen.

52137 Überprüfung, ob BE$(WZ) »UND« ist. Wenn ja, wird UD auf »1« gesetzt.

52140 IC ist »0«. Es steht fest, daß BE$(WZ) in keiner Tabelle oder im gesamten Wortschatz vertreten ist. Es erfolgt eine entsprechende Fehlermeldung.

52150 Die Wortzählvariable wird auf das nächste Wort gesetzt.

52160 Überprüfung, ob der Befehlssatz die zulässige Wortzahl von maximal zehn Worten überschreitet. Enthält das neue Wort BE$ (WZ) Zeichen? Steht UD auf »1«? Trifft eine dieser Bedingungen zu, wird der Codiervorgang:
a) beendet, wenn der Befehlssatz zu Ende ist (BE$(WZ)=»«),
b) unterbrochen, wenn ein Teilbefehl des Befehlssatzes BE$ ausgeführt werden muß (UD=1).

52170 Die Check-Variable IC wird auf »0« gesetzt. Der Lesevorgang beginnt erneut.

Ab **60000** Definieren der Worttabellen.

Starten Sie »LISTING 6« mit RUN. Sie werden mit »WAS NUN?« aufgefordert, einen Befehlssatz einzugeben. Verwenden Sie dazu Wörter, die im Wortschatz enthalten sind. Drücken Sie anschließend <RETURN>. Das Programm gibt die aus der Codierung resultierenden Zahlen aus:
Verbzahl **VE**
1. Gegenstandszahl **G1**
2. Gegenstandszahl **G2**
Objektzahl **OB**
Personenzahl **PE**

Geben Sie zur Übung verschiedene Befehlssätze ein (wahlweise mit mehreren Befehlen). Untersuchen Sie die Ergebnisse. Sie können Beispielssätze, die wir bereits theoretisch codiert haben, eingeben und die Ergebnisse mit den im Kurs besprochenen vergleichen. Ergibt der Vergleich keine Unterschiede, arbeitet das Modul tadellos.

Vom Sinn der Variablen und Zahlen

Bevor ich das Geheimnis lüfte, was wir mit den Verb-, Objektzahlen usw. anfangen werden, müssen wir uns einem kleinen Problem zuwenden.

Variable	Funktion
AZ	Wortanzahl gem. Tabelle der auszusortierenden Wörter
AU$(1) bis AU$(AZ)	Tabelle der auszusortierenden Worte
VZ	Anzahl der Verben
VE$(1) bis VE$(VZ)	Tabelle der Verben
GZ	Anzahl der Gegenstände
GE$(1) bis GE$(GZ)	Tabelle der Gegenstände
OZ	Anzahl der Objekte
OB$(1) bis OB$(OZ)	Tabelle der Objekte
PZ	Anzahl der Personen
PE$(1) bis PE$(PZ)	Tabelle der Personen
UD	Variable für »UND«
BE$	String, der den kompletten Befehlssatz enthält
BE$(1) bis BE$ (maximal 10)	Einzelne Worte des Befehlssatzes BE$, überflüssige Worte sind aussortiert
WZ	Wortzählvariable (gibt an, welches aktuelle Wort bearbeitet wird)
IC	Checkvariable
I	Schleife
I1	Schleife, innerhalb der Schleife I
VE	Verbzahl (wird aus Verbtabelle ermittelt)
OB	Objektzahl (wird aus Objekttabelle ermittelt)
PE	Personenzahl (wird aus Personentabelle ermittelt)
G1, G2	1. und 2. Gegenstandszahl (werden aus Gegenstandstabelle ermittelt)

(Variablen, die mit »I« beginnen, sind entweder Check-Variablen oder Schleifen)

Tabelle 2. Verwenden Sie diese Variablen in eigenen Abenteuerspielen

Betrachten Sie die folgenden beiden Sätze:
1. VERLIER DAS SEIL.
2. VERLIERE DAS SEIL.

Für uns ist hier zunächst kein Unterschied zu sehen. Bei genauerem Hinsehen stellen wir fest, daß das Verb im ersten Satz »VERLIER« und im zweiten »VERLIERE« lautet. Der Sinn ist eindeutig. Dem Computer müssen Sie klarmachen, daß beide Worte dasselbe bedeuten.

Statt »VERLIER« und »VERLIERE« könnte unser Beispiel auch »SAG« und »SAGE« lauten. Das Problem ist lösbar. Bisher lief die Verb-Untersuchung im Programm (Zeile 52025 bis 52050) wie folgt ab:

Wir haben überprüft, ob das aktuelle Befehlswort BE$(WZ) in der Verbtabelle enthalten ist:

```
IF BE$(WZ) = VE$(I) THEN VE=I : IC = 1
```

Die Variable »I« durchläuft die Werte 1 bis VZ, wobei VZ die Anzahl der Verben darstellt.

Wir müssen diese Abfrage ändern. Es soll nicht mehr überprüft werden, ob BE$(WZ) mit VE$(I) identisch ist, sondern ob das Verb VE$(I) das Suchkriterium BE$(WZ) enthält. Die Programmiertechnik dieser Überprüfung haben Sie bereits im Kapitel über Stringoperationen gelernt (lesen Sie diesen Abschnitt gegebenenfalls nochmal nach).

Schreiben Sie die Abfrage (Programmzeile 52040) folgendermaßen um:

```
52040 :IFBE$(WZ)=LEFT$(VE$(I),LEN(BE$(WZ)))THENVE=
I:IC=1
```

Starten Sie das Programm erneut und überprüfen Sie, ob es ordnungsgemäß abläuft. Trifft dies zu, müßte für VERLIER und VERLIERE eine identische Verbzahl und keine Fehlermeldung entstehen. Geben Sie lediglich den Buchstaben »V« ein, erhalten Sie ebenfalls die Verbzahl für das ganze Wort »VERLIERE«. Jeder Verb-Befehl kann damit nach Belieben abgekürzt werden.

Schnell führen derartige Abkürzungen zu Mißverständnissen. Sind in der Verbtabelle zwei Verben enthalten, die mit den gleichen Buchstaben beginnen (z.B. »SAGE« und »SUCHE«), kann das Programm bei der Eingabe des Buchstaben »S« nicht wissen, welcher Befehl gemeint ist. Es nimmt in einem derartigen Fall die größte Verbzahl. Diese Tatsache braucht uns nicht zu stören. Der Spieler wird rasch herausfinden, welche Befehle er abkürzen kann.

Verbfamilien

Unser Befehlsanalyse-Programm ist inzwischen sehr gut geworden, aber wir wollen einen Schritt weitergehen.

Betrachten wir die folgenden beiden Verben: NIMM und NEHME.

Im Prinzip besteht zwischen den Befehlssätzen NIMM DAS SCHWERT und NEHME DAS SCHWERT kein Unterschied. Der Computer sieht das nicht so. Bevor wir uns mit der Lösung dieses neuen Problems beschäftigen, wollen wir Ihnen ein anderes Beispiel vor Augen führen.

Nehmen wir an, unser Spieler hat ein Brett und ein Seil. Er will das Seil am Brett festbinden. Folgende Befehlseingaben könnte er ausprobieren:
1. Binde das Seil am Brett fest.
2. Befestige das Seil am Brett.
3. Verknote das Seil am Brett.
4. Verbinde das Seil mit dem Brett.
5. Mache das Seil am Brett fest.
6. Binde das Seil ans Brett.

Diese Sätze sind Musterbeispiele dafür, warum wir zwei Gegenstandsvariablen G1 und G2 für die Befehlsanalyse reserviert haben. In allen sechs Eingabefolgen kommen zwei Gegenstände vor: Seil und Brett.

Die im Satz auftauchenden Gegenstände sind im jedem Fall identisch. Filtern wir die Verben VE$ und die auszusortierenden Worte (AU$) heraus:
1. VE$: BINDE AU$: das, am, fest
2. VE$: BEFESTIGE AU$: das, am
3. VE$: VERKNOTE AU$: das, am
4. VE$: VERBINDE AU$: das, mit, dem
5. VE$: MACHE AU$: das, am, fest
6. VE$: BINDE AU$: das, ans

Wir stellen mehrere, sinngemäß gleiche Verben fest – eine Verbfamilie. Eingabesatz 5 wollen wir verbieten. In ihm ist kein eindeutiges Verb vorhanden. Zwar kann »MACHE« als Verb angesehen werden, aber es benötigt ein Bezugswort, z.B. »fest« (»MACHE FEST«). Das Wort »fest« kann jedoch nicht als Ersatz für ein Verb dienen, denn in Satz 1 wird es als auszusortierendes Wort betrachtet. Vermeiden Sie grundsätzlich das Befehlsverb »MACHE«. Bei jeder Eingabe dieses Verbs muß der Spieler die Meldung »ICH KENNE DAS WORT MACHE NICHT« vom Programm erhalten. Dazu ist es nicht erforderlich, diese Fehlermeldung eigens zu programmieren. Unser Befehlsanalyse-Programm bringt den Hinweis für jedes Wort, das nicht in einer der Worttabellen vertreten ist (einschließlich der Tabelle der auszusortierenden Wörter). Unser Problem ist, daß die Befehlsanalyse sich nicht nur auf einzelne Verben, sondern auf ganze Verbfamilien beziehen muß. Die Verben NIMM und NEHME sind ein Beispiel dafür.

Eine einfache Lösung dafür wäre:

Wir erweitern die Verbtabelle VE$ kurzerhand um das Verb »NEHME«. Damit würde es eine eigene Verbzahl zugeordnet erhalten. Die Folge wären unterschiedliche Verbzahlen für »NIMM« und »NEHME«. Diese Tatsache müßte man im Action-Modul berücksichtigen. Wollte dieses feststellen, ob der Spieler etwas nehmen will, dann würde die Abfrage so aussehen:

```
IF VE = A OR VE = B THEN:REM:
SPIELER WILL ETWAS NEHMEN.
```

»A« steht für die Verbzahl von »NIMM«, »B« für das Wort »NEHME«.

Diese Lösung würde für unser erstes Beispiel gut funktionieren. Beim zweiten Eingabesatz (Spieler will Seil mit Brett verbinden) wäre eine lange IF-THEN-Abfrage nötig. Diese Lösung frißt außerdem viel vom kostbaren RAM-Speicher.

Die Ideallösung ist: »NIMM« und »NEHME« erhalten dieselbe Verbzahl. Die im Action-Modul notwendige Abfrage gestaltet sich wesentlich unkomplizierter:

```
IF VE = A THEN:REM: SPIELER WILL ETWAS NEHMEN.
```

Wie programmiert man diese Musterlösung?

Die eine Möglichkeit wäre, mehrere Alternativ-Verbtabellen zu erstellen. In der normalen Verbtabelle stände an erster Stelle das Verb »NIMM«, in einer alternativen Tabelle »NEHME«. Betrachten wir aber unsere zweite Wortbeispieltabelle, stellen wir fest, daß dies ein erheblicher Aufwand wäre. Unsere zweite Verbfamilie besteht aus vier Mitgliedern (binde, befestige, verknote, verbinde), die erste lediglich aus zwei Worten (nimm, nehme). Wir müßten mehrere Alternativ-Tabellen programmieren. An Speicherplatz hätten wir nichts gespart, außerdem würde sich die Programmierung kompliziert gestalten. Es geht bedeutend einfacher (man muß nur wissen, wie):

Das Verb »NIMM« steht in unserer Verbtabelle in der DATA-Zeile 60060.

```
60060 DATA NIMM, VERLIERE, OEFFNE, GIB, SAGE,
INVENTUR, BEFESTIGE
```

Ändern wir diese Zeile:

```
60060 DATA NIMM, NEHME1, VERLIERE, OEFFNE, GIB,
SAGE, INVENTUR, BEFESTIGE
```

Wir haben das Verb »NEHME« direkt hinter »NIMM« eingefügt. Halt, das stimmt nicht ganz. Das neue Verb heißt »NEHME1« – mit der Zahl »1«, die sich an die Buchstaben anschließt. Dies ist kein Druckfehler.

Was soll der »Einser«, werden Sie mit Recht fragen. Stören kann er nicht. Wenn wir das Programm starten und »NEHME« eingeben, wird es akzeptiert. Der Computer betrachtet »NEHME« als Abkürzung für das in seiner Verbtabelle an zweiter Stelle gespeicherte Verb »NEHME1«.

Soeben fällt mir auf, daß mir ein Fehler unterlaufen ist. Haben Sie ihn auch entdeckt? Ich habe vergessen, die VZ-Variable in Zeile 60070 zu ändern. Ein zusätzliches Verb wurde in die Tabelle eingebaut. Statt sieben Verben sind es nun acht. Diese Änderung wollen wir nachholen:

```
60070 VZ=8...
```

Solche Fehler sind tückisch. Sie werden selten im Programm wirksam und sind daher schwer zu finden. Denken Sie stets daran: Fügen Sie irgendeiner Worttabelle ein weiteres Wort hinzu, muß die entsprechende Variable für die Anzahl der Worte in dieser Tabelle korrigiert werden.

Zurück zur Frage: »Warum NEHME1«?

Stammverben haben Vorrang

Voraussetzung für das Funktionieren dieser Ergänzung ist die beschriebene Änderung von Programmzeile 52040. Können Sie sich an den VAL-Befehl erinnern, den wir bei den Stringoperationen besprochen haben?

Nach der Änderung der Verbtabelle sieht diese folgendermaßen aus:

```
VE$(1) = "NIMM"
VE$(2) = "NEHME1"
VE$(3)... etc.
```

Probieren Sie aus, welchen Wert VAL(VE$(2)) ergibt. Geben Sie im Direktmodus ein:

```
PRINT VAL("NEHME")
```

Welches Ergebnis erscheint, wenn Sie anschließend <RETURN> drücken? Richtig, »0«. Prüfen Sie es nach. Haben Sie als Ergebnis »1« erwartet, sollten Sie den Abschnitt über VAL im Handbuch des C64 noch einmal lesen.

Geben Sie folgende Zeile ein:

```
PRINT VAL(RIGHT$("NEHME1",1))
```

Jetzt erhalten Sie nach Drücken der RETURN-Taste das erhoffte Ergebnis: »1«. Setzen wir für »NEHME1« den String »NIMM« ein, bekommen wir als Resultat »0«. Es ist gleichgültig, ob wir »NIMM« oder »NIMM0« schreiben.

Folgendes läßt sich feststellen:

Das Stammverb steht immer vor den alternativen Verben in der Tabelle. Die alternativen Verben folgen direkt nach dem Stammverb und enden mit einer Zahl.

Betrachten wir die Verbfamilie BINDE, BEFESTIGE, VERKNOTE, VERBINDE. In eine Verbtabelle eingebaut, würde diese Worttabelle so aussehen (wobei es egal ist, welches Mitglied der Familie als Stammverb gewählt wird):

```
VE$(1) = "BINDE"
VE$(2) = "BEFESTIGE1"
VE$(3) = "VERKNOTE2"
VE$(4) = "VERBINDE3"
```

Bei der Numerierung bin ich nicht von unserer bisherigen Verbtabelle ausgegangen. Unser Ziel besteht darin, für jedes Mitglied einer Verbfamilie die gleiche Verbzahl VE zu erhalten. Ich lege fest, daß die Verbzahl für jedes Verb der »Familie« identisch mit dem Wert ist, den man für das Stammverb der Tabelle bekommt.

Wissen Sie noch, wie wir unsere Verbzahl bisher erhalten haben? Eine Schleife »I« durchlief die VE$-Tabelle von 1 bis VZ (Anzahl der Verben). Dabei haben wir überprüft, ob das aktuelle Befehlswort BE$(WZ) mit VE$(I) identisch war. Stimmten beide Strings überein, so galt: VE = I (Verbzahl = augenblicklicher Stand der Schleife).

Im Programm sah dies in der ersten Version so aus:

```
52030 FORI = 1 TO VZ
52040 :IFBE$(WZ) = VE$(I) THEN VE=I
52050 NEXT I
```

Dieses Programm habe ich möglichst vereinfacht. Wenn wir unsere Verbtabelle durchlaufen lassen, erhalten wir im Moment noch für jedes Verb eine individuelle Verbzahl. Wie stellen wir es an, daß für jedes Wort einer Verbfamilie dieselbe Verbzahl VE gilt?

Bisher war VE vom jeweiligen Stand der Schleifenvariable I abhängig, bei dem sich BE$(WZ) mit VE$(I) als identisch erwies. Wir müssen einen Weg finden, aus dessen Verlauf sich VE in Abhängigkeit von der Schleifenvariablen I und vom VAL-Wert des jeweiligen Verbs VE$(I) ergibt. Dazu sollten wir die neue Verbtabelle betrachten:

Für das Stammverb »BINDE« erhalten wir den VAL-Wert »0«.

Für das erste alternative Verb »BEFESTIGE« ist der VAL-Wert »1«, für das zweite entsprechend »2« und schließlich »3« für das dritte Ersatzwort.

Die Endziffer im String gibt die Position des Alternativverbs ab Stammverb an. Es wurde festgelegt, daß die Verbzahl für alle Mitglieder einer Verbfamilie identisch mit dem Wert des Stammverbs der Familie sein soll. Dieses besitzt den VAL-Wert »0«. Die Lösung unseres Problems liegt klar auf der Hand:

```
52030 FOR I = 1 TO VZ
52040 :IFBE$(WZ) = VE$(I) THEN VE =
      I-VAL(RIGHT$(VE$(I),1))
52050 NEXTI
```

Wir ziehen von dem I-Wert, der die Identität festgestellt hat, kurz und bündig den VAL-Wert ab. Subtrahieren wir vom sich für das Stammverb ergebenden I-Wert den VAL-Wert des Stammverbs, erhalten wir den Verbzahl-Wert VE des Stammverbs (der VAL-Wert des Stammverbs ist immer »0«).

Wenn wir vom I-Wert für das erste Alternativverb den VAL-Wert dieses Ersatzwortes abziehen, bekommen wir dieselbe Verbzahl VE. Entsprechend gilt dies für die anderen Alternativverben. Wir haben unsere angestrebte Ideallösung gefunden. Das Wichtigste dabei ist, daß die Befehlsanalyse kaum länger geworden ist.

Alternative Verben

Vergessen wir dabei nicht, die Programmzeile 52040 erneut zu ändern. Der Vergleich zwischen BE$(WZ) und VE$(I) muß dergestalt erfolgen, daß überprüft wird, ob BE$(WZ) in VE$(I) enthalten ist. Denn nur dann werden Abkürzungen für Verben akzeptiert. Dies ist die Grundbedingung, die es uns erlaubt, Strings der Alternativverben mit Nummern zu versehen.

Die endgültige Zeile 52040 lautet:

```
52040 :IFBE$(WZ)=LEFT$(VE$(I),LEN(BE$(WZ)))THENVE=
      I-VAL(RIGHT$(VE$(I),1)) :IC=1
```

Es ist geschafft: Unser Analysemodul ist fertig. »LISTING 7«, das Sie von der beiliegenden Diskette laden sollten, bietet Ihnen eine komplette Befehlseingabe mit unserer idealen Befehlsanalyse in einem Unterprogramm, das ab Zeile 50000 steht. Das Programm ist komprimiert (soweit dies möglich war). Dies ist die Grundbasis für die folgenden Beispielprogramme. Das Modul kann für jedes Adventure verwendet werden. Es enthält keine programmspezifischen Formulierungen. Änderungen lassen sich problemlos durchführen.

Das Modul kann nur dann arbeiten, wenn vorher im Programm die Worttabellen mit den entsprechenden Variablen definiert wurden. Beachten Sie dabei, daß die Verbtabellen grundsätzlich die längstmöglichen Wörter enthalten (z.B. »VERLIERE« und nicht »VERLIER«). Nur dann ist die Möglichkeit einer Abkürzung der Verben gewährleistet. Prinzipiell könnte man für andere Worttabellen (z.B. Objekte, Gegen-

stände usw.) Abkürzungen erlauben. Ich halte das für überflüssig.

Laden Sie »LISTING 7« in den Computer, LISTen es und betrachten sich die die folgenden Zeilen:

```
50250 FORI=1TOVZ:IFBE$(WZ)=VE$(I)THENVE=I:IC=1
50251 IFLEN(BE$(WZ))<3THEN50260
```

Die beiden IF-THEN-Abfragen sorgen dafür, daß Abkürzungen für Verben mindestens drei Buchstaben haben müssen. Wörter mit ein bis zwei Buchstaben werden jedoch akzeptiert, wenn sie in der Verbtabelle enthalten sind (z.B. Richtungsangaben wie N, S, O, W, NW usw.).

Die Spielkarte

Als Resultat des bisherigen Kursverlaufs steht uns ein komfortables Befehlsanalyse-System zur Verfügung. Wir sind dem optimalen Abenteuerspiel bereits sehr nahe gekommen. Die letzten Kursabschnitte enthielten eine Menge Theorie. Damit haben wir aber die wesentlichste Denkarbeit hinter uns gebracht: Sie haben gelernt, Tabellen zu definieren und mit Strings zu arbeiten. Als Entschädigung dafür wird in den folgenden Kursteilen der theoretische Teil bedeutend knapper ausfallen.

Das Hauptproblem, mit dem sich jeder Adventure-Programmierer herumschlagen muß, ist das Umsetzen der Spielkarte in ein für den Computer verständliches Programm. Wer ein Adventure-Listing in Basic unter die Lupe nimmt, wird auf den ersten Blick verwirrt sein: Ein Rattenschwanz von Variablen und GOTO/GOSUB-Anweisungen leisten ihren Beitrag zur Unübersichtlichtkeit. Es erscheint nahzu unmöglich, den Programmablauf vernünftig zu verfolgen. Eine weitere Tatsache ist, daß in kaum einem anderen Listing so viele Fehler auftauchen wie in Abenteuerspielen. Das liegt vor allem daran, daß sich viele Adventure-Programmierer nach gewisser Zeit in ihrem eigenen Machwerk nicht mehr zurechtfinden. Die Programmierung war leider zu »chaotisch«.

Folgende Eigenschaften müssen wir von unserem System zur Programmierung der Spielkarte verlangen:

1. Sehr hohe Übersichtlichkeit – eventuell auftretende Fehler müssen leicht behoben werden können.
2. Möglichst geringer Speicherplatzverbrauch – je besser das System zur Programmierung der Spielkarte ist, desto mehr Räume lassen sich unterbringen.
3. Es muß lediglich eine Variable verändert werden, um das Spiel an jeder beliebigen Stelle (Raum) starten zu lassen – zum späteren Austesten des Spiels unverzichtbar.

Vom Drehbuch zum Drehort

Im Prinzip gibt es verschiedene Verfahrensweisen, Mängel besitzen sie alle. Nach zeitaufwendigen Versuchen hat sich bei mir eine Technik bewährt, die ich Ihnen vorstellen möchte. Mit dem mageren Basic 2.0 des C 64 ist diese Technik jedoch nicht realisierbar.

Modifiziertes Utility für Zeilensprünge

Unser erster Schritt soll zunächst darin bestehen, das bescheidene Basic des C 64 um drei wichtige Befehle zu erweitern, bzw. drei bereits vorhandene Betriebssystem-Routinen für unsere Zwecke zu modifizieren.

Laden Sie »LISTING 8« von der beiliegenden Diskette und starten das Programm mit RUN. Ab sofort stehen uns folgende neue, bzw. erweitere Befehle zur Verfügung:

```
GOTO Zeilennummer
GOSUB Zeilennummer
RESTORE Zeilennummer
```

Diese bekannten Basic-Anweisungen wurden dergestalt »restauriert«, daß sie im Gegensatz zur Normalfunktion auch in Abhängigkeit von einer Variablen oder eines Formelausdrucks eingesetzt werden können. Die ON GOTO- und ON GOSUB-Anweisungen können Sie ab sofort getrost »vergessen«. Folgende Programmiermöglichkeiten ergeben sich durch das Utility (drei Beispiele):

```
 10 REM DEMOPROGRAMM 1
 20 X = 100
 30 GOTO X
100 PRINT"SPITZE !"
110 END

 10 REM DEMOPROGRAMM 2
 20 GOSUB 90+10
 30 END
100 PRINT"SPITZE !" : RETURN

 10 REM DEMOPROGRAMM 3
 20 INPUT"WERT"; X
 30 IF X<1 OR X>3 THEN GOTO 20
 40 RESTORE 100*3 : READX$ : PRINT X$ :GOTO 20
100 DATA HALLO
200 DATA WIE
300 DATA GEHTS
```

Vor allem beim dritten Beispielprogramm könnte dem einen oder anderen **die** Idee zur Programmierung der Spielkarte kommen. Zuvor müssen wir allerdings folgenden Aspekt berücksichtigen: Durch die drei modifizierten Befehle bietet sich eine völlig neue Programmiermöglichkeit. Voraussetzung dazu ist, daß die Programmodule und Tabellen (DATAs) in vorher exakt definierten Programmzeilenbereichen stehen müssen. Eine beispielhafte Aufteilung zeigt Ihnen Bild 17.

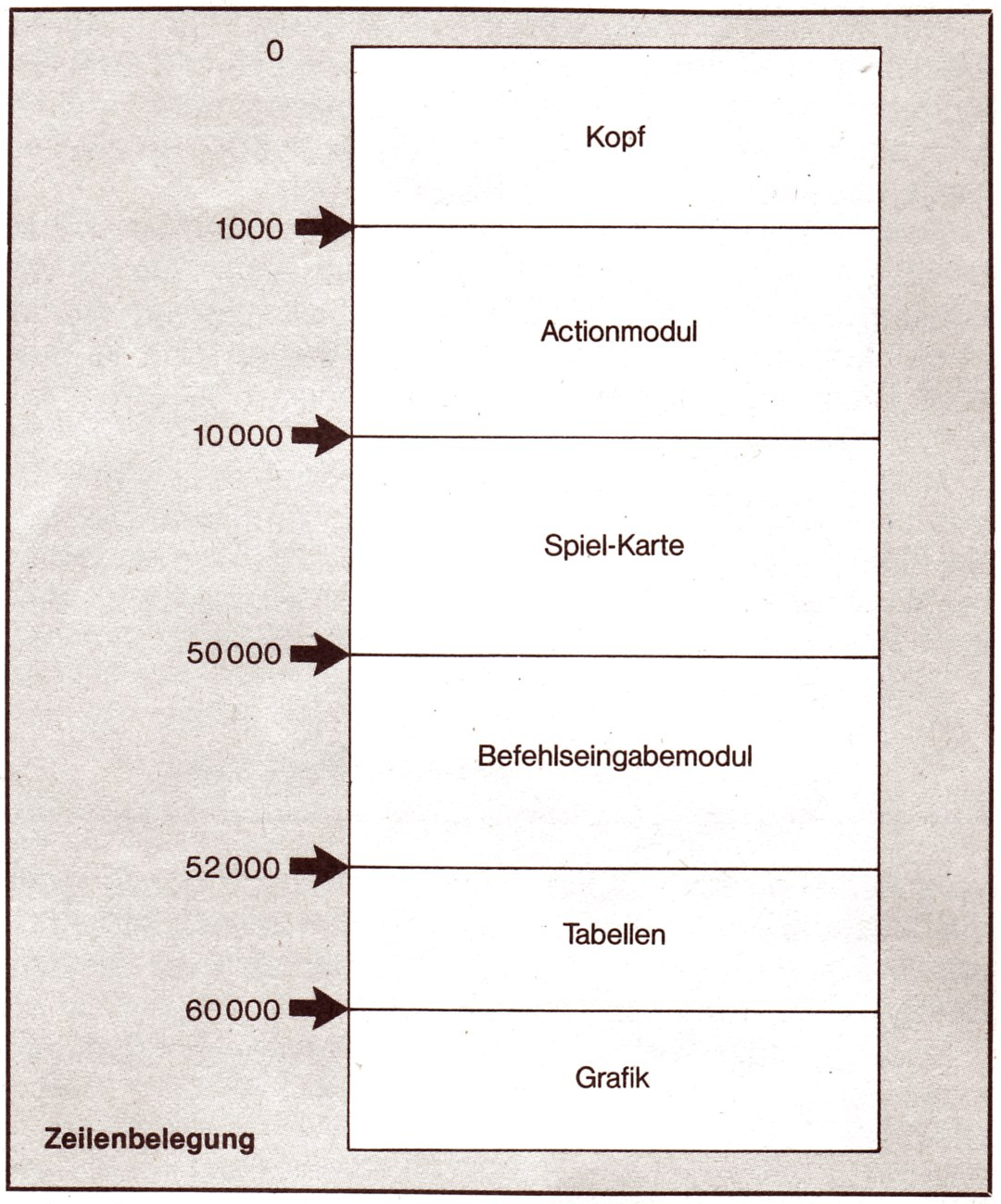


Bild 17. Sauber und übersichtlich: die ideale Programmaufteilung eines Basic-Adventures

Die Spielkarte (Räume) befindet sich in den Programmzeilen ab 10000. Innerhalb dieses Bereichs läßt sich die Unterteilungsmethode fortsetzen:

Raum 1 Zeilen 10100 bis 10120
Raum 2 Zeilen 10200 bis 10220 usw.

Dieses Programm ist noch nicht lauffähig, da jegliche Steuerung fehlt. Das wäre die Aufgabe des Action-Moduls

	1	2	3	4	5	6	7	8	9	10
RI	N	S	W	O	NW	NO	SW	SO	RAUF	RUNTER

Bild 18. Aufbau der Feldvariablen RI (Richtungen)

(Zeilen 1000 bis maximal 9999). Wir wollen zunächst eine primitive Steuerung simulieren:

```
1000 REM ACTION-MODUL
1010 INPUT"IN WELCHEN RAUM WOLLEN SIE GEHEN.";ZN
1015 IF ZN<1 OR ZN>6 THENPRINT"DAS GEHT NICHT":
     GOTO1010
1020 GOSUB 10000+ZN*100
1030 GOTO 1000
```

Interessant ist Zeile 1020. Dort steht die Formel, durch die mit GOSUB der aktuelle Raum erreicht werden kann. Dieses Action-Modul ist extrem simpel. Sie können nach Belieben jeden Raum direkt anlaufen – falls man überhaupt von »Laufen« reden kann.

Löschen Sie folgende Zeilen: 1010, 1015, 1020 und 1030. Merken Sie sich die Formel zur Errechnung des jeweiligen Raumes: **Raum = 10000+ZN*100**. ZN ist die jeweilige Raumnummer. Sie sollte mit den Zahlen auf dem gezeichneten Spielplan identisch sein. Die Formel bestimmt die Zeilennummer, ab der der Raum im Programm zu finden ist. Es ist an der Zeit, unser Befehlseingabemodul in Bewegung zu setzen. Das Action-Modul ist so zu ändern, daß man Räume nicht mehr direkt ansteuern kann. Der Spieler sollte lediglich von einem Raum in den anderen gelangen können. Dazu ist erneut etwas Theorie notwendig.

Programmierung der Adventure-Tabellen

Erlauben wir dem Spieler unseres Adventures, direkte Himmelsrichtungen einzugeben, um sich zu bewegen. Will er z.B. nach Norden, genügt die Eingabe von »N«. Manche Adventures verlangen Befehle wie »GEHE NORD«. Dies ist unnötig und wird dem Spieler schnell auf die Nerven fallen. Wesentlich bequemer ist es, ein bis zwei Buchstaben zur Bewegung im Abenteuerspiel benutzen zu können. Sie erinnern sich bestimmt daran, was ich im Abschnitt zur Programmierung des ersten Moduls festgelegt habe: Himmelsrichtungen sollen wie gewöhnliche Verben behandelt werden. Aus diesem Grund wollen wir die zehn möglichen Bewegungsrichtungen in unsere Verbtabelle aufnehmen. Es ist zugleich der Beginn der Programmierung einzelner Worttabellen.

Ergänzen Sie unser Beispielprogramm (Action-Modul) um folgende Basic-Zeilen:

```
52000 REM TABELLEN
52005 RESTORE 52000
52010 REM VERBTABELLE---------------------
52020 DATA N,S,W,O,NW,NO,SW,SO,RAUF,RUNTER
52100 VZ=10:DIMVE$(VZ):FOR I=1TOVZ:READ VE$(I):NEXT
53000 RETURN
```

Legen Sie sich ein Blatt Papier an, auf dem Sie die folgenden Tabellen ausführlich notieren – den jeweiligen Wert zu jedem einzelnen Verb, Objekt, Gegenstand usw. Zu Beginn des Beispielprogramms muß der Aufruf des Tabellenunterprogramms eingebaut werden:

```
30 GOSUB 52000 : REM TABELLEN DEFINIEREN
```

Wir erhalten damit Verbzahlen VE für die einzelnen Himmelsrichtungen. Bevor aber die GEH-Routine in das Action-Modul eingebaut werden kann, müssen wir die Spielkarte verbessern. Sie besteht bislang nur aus knappen Texten. Wir sollten Informationen einbauen, die angeben, wie die Räume miteinander verbunden sind. Wie dies funktioniert, zeigt Ihnen »LISTING 9«, das Sie von der beiliegenden Diskette laden und sich ansehen sollten. Jeder der sechs Räume benötigt eine zusätzliche DATA-Zeile mit zehn numerischen Werten. Diese sollen dem C64 mitteilen, ob und in welche Rich-

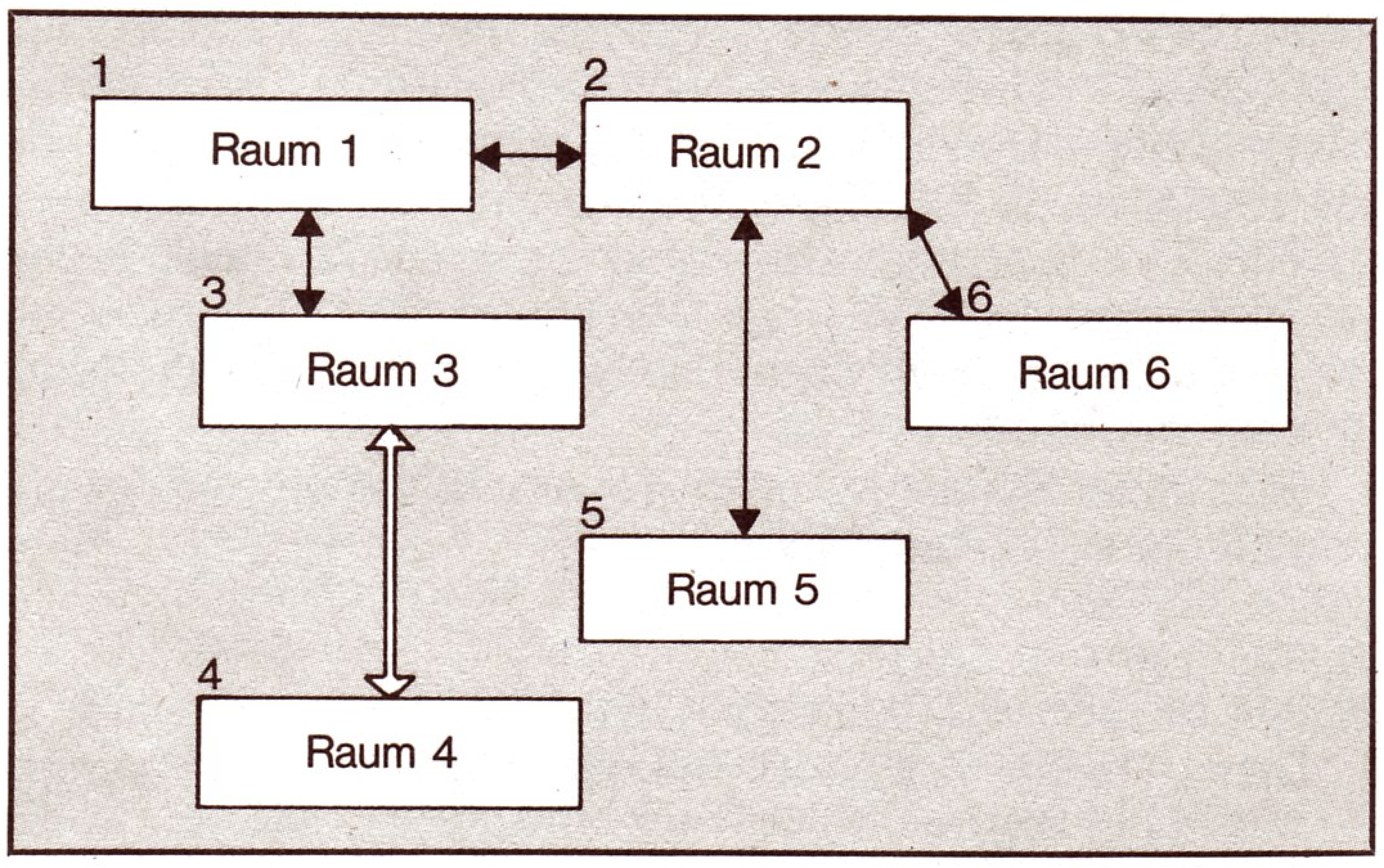


Bild 19. Beispielkarte zur Programmierung der Richtungen

tungen gelaufen werden kann. Immer, wenn der Spieler einen Raum betritt, müssens diese zehn Zahlen in das Variablenfeld RI(1) bis RI(10) eingelesen werden (Bild 18).

Das bisher entstandene Beispielprogramm in Basic belegt wenige Kopfzeilen (Befehlserweiterung für GOTO X usw., Befehlseingabemodul ab 50000). Die Zeilen 10000 bis maximal 49999 sind für die Spielkarte reserviert. Wir haben demnach 39999 Zeilen für die Programmierung der Spielkarte zur Verfügung. Man kann davon ausgehen, daß zur programmtechnischen Definition eines einzelnen Raumes der Spielkarte (Text und Action) maximal 100 Zeilen benötigt werden. In 39999 Zeilen ließen sich 400 Räume unterbringen. Das ist lediglich ein theoretischer Wert. Ein vernünftiges Adventure wird nie 400 Räume besitzen, außerdem ist dies mit 38 KByte RAM-Speicher nicht zu vereinbaren. Trotzdem war ich großzügig und habe 39999 Zeilen für ein Maximum von 400 Räumen zur Verfügung gestellt.

Für unsere weiteren Überlegungen verwenden wir die Beispielkarte in Bild 19. Sie finden darauf sechs Räume, durchnumeriert von 1 bis 6. Ich habe absichtlich keinen größeren Spielplan gewählt, da er für unsere ersten Versuche vollkommen ausreicht. Ergänzen Sie das bisherige Programm (Basic-Erweiterungsroutine und Befehlseingabemodul) um folgende Zeilen:

```
10102 DATA 0,3,0,2,0,0,0,0,0,0
10202 DATA 0,5,1,0,0,0,0,6,0,0
10302 DATA 1,0,0,0,0,0,0,0,0,4
10402 DATA 0,0,0,0,0,0,0,0,3,0
10502 DATA 2,0,0,0,0,0,0,0,0,0
10602 DATA 0,0,0,0,2,0,0,0,0,0
```

Diese Zeilen finden Sie als »LISTING 10« auf der beiliegenden Diskette. Folgendes Beispiel soll die Funktion dieser Daten verdeutlichen:

Der Spieler steht im Raum A. Im Variablenfeld RI(1) bis RI(10) ist festgelegt, welche anderen Räume unter welchen Richtungen erreicht werden können. Nehmen wir an, es gibt nur eine mögliche Richtung, um Raum A zu verlassen – Süden (S). In dieser Richtung kommt man in Raum B. Die DATA-Zeile für Raum A müßte in diesem Fall so aussehen:

```
DATA 0,B,0,0,0,0,0,0,0,0
```

In der Praxis wird statt »B« der Wert des Raumes eingesetzt. Bedingt durch die DATA-Zeile, müßte das RI-Feld so aussehen:

```
RI(1)=0, RI(2)=B, RI(3) bis RI(10)=0
```

Ein anderes Beispiel: Der Spieler will Raum A in Richtung Norden verlassen. Auf die Frage »WAS NUN?« gibt er »N« ein. Das Befehlseingabemodul gibt den Wert für das eingegebene Verb »N« aus (VE=1). Nun tritt das Action-Modul in Erscheinung.

Es überprüft, ob der Verbzahlwert VE im Bereich zwischen 1 und 10 liegt – ob der Spieler sich bewegen will oder nicht

(dann ist VE größer als »10«). Das Action-Modul stellt fest, daß der Spieler fort möchte und macht weiter. Es prüft jetzt, ob die vorgegebene Bewegungsrichtung existiert. Außerdem muß es feststellen, ob RI(VE) den Wert »0« hat oder einen anderen. Ist RI(VE) = 0, erscheint die Fehlermeldung »Kein Weg in diese Richtung!«. Ist RI(VE) < > 0 (in Raum A trifft dies zu, wenn sich der Spieler nach Süden bewegen will), fährt das Modul in seiner Arbeit fort.

Gibt unser Spieler »S« statt »N« ein, registriert das Action-Modul die neue Raumnummer:

```
ZN = RI(VE)
GOSUB 10000 + ZN*100
Neues Feld RI von Raum B einlesen.
```

Diese Theorie setzen wir mit den folgenden Ergänzungszeilen in die Praxis um:

```
1010 GOSUB 50000 : REM BEFEHLSEINGABEMODUL
1100 REM GEHEN IN EIN NEUES ZIMMER
1105 IFVE<1 OR VE>10 THEN1200
1110 IF RI(VE)=0 THENPRINT"KEIN WEG IN DIESE
     RICHTUNG !" :GOTO1200
1120 ZN = RI(VE) : PRINT "<CLR/HOME>"
1130 GOSUB 10000 + ZN*100
1140 RESTORE 10000 + ZN*100
1150 FOR I=1 TO 10 : READ RI(I) : NEXT
1200 GOTO 1000
```

Wir müssen im Programmkopf noch eine Zeile einfügen. Darin wird bestimmt, in welchem Raum sich der Spieler zu Beginn des Spiels befindet:

```
100 ZN = 1 : GOTO 1130
```

Das Spiel beginnt demnach in Raum 1. Soll es in irgend einem anderen Raum anfangen, muß der Wert in Zeile 100 entsprechend geändert werden. Beim Austesten oder bei der Fehlersuche ist dies eine große Hilfe.

Das Ziel dieses Kursabschnittes ist erreicht - Sie wissen nun, wie man eine Spielkarte programmiert. Die vorgestellte Version ist mit einem spärlichen Raumbeschreibungstext ausgestattet. Dies können Sie leicht selbst verbessern - das nötige Wissen dazu haben Sie jetzt. Den gesamten Verlauf dieses Kursteils mit den erarbeiteten Listings finden Sie als Gesamtprogramm in »LISTING 11« auf der beiliegenden Diskette. Hier handelt es sich bereits um ein Adventure, wenn auch von der simpelsten Art. Alles, was Sie machen können, ist in den Räumen herumlaufen.

Was unabdingbar fehlt, ist zusätzlich zur Raumbeschreibung eine Bildschirmausgabe der Richtungen, in die man sich bewegen kann. Wir programmieren dazu eine Routine, die nach jeder Raumbeschreibung den Hinweis »MÖGLICHE RICHTUNGEN:« ausgibt. Die entsprechenden Basic-Zeilen sind in »LISTING 12« auf der beiliegenden Diskette abgespeichert. Da es sich um eine bereits mehrmals besprochene FOR-NEXT-Schleife handelt, möchte ich auf eine ausführliche Dokumentation verzichten. Die Programmierung der Spielkarte ist nahezu abgeschlossen.

Gegenstands- und Objekttabellen definieren

Wir brauchen noch zwei Programmroutinen. Eine davon soll bei jeder Raumbeschreibung die sich darin befindlichen Gegenstände ausgeben. Der Adventure-Wortschatz muß um einige Gegenstände erweitert werden. Im Variablenfeld GE$(1) bis GE$(GZ) sind die Namen der Gegenstände abgelegt, in GZ die Anzahl. Wir benötigen ein weiteres Variablenfeld. Darin muß angegeben sein, in welchen Räumen sich die Gegenstände befinden: GE(1) bis GE(GZ).

Wir ergänzen unser Programm um »LISTING 13« (Sie finden es unter demselben File-Namen auf der beiliegenden Diskette):

```
52200 REM GEGENSTANDSTABELLE-------
52210 DATA SCHWERT,1
52211 DATA SEIL,2
52212 DATA SCHLUESSEL,4
52213 DATA DIAMANT,5
52300 GZ=4:DIM GE$(GZ):DIM GE(GZ):FORI=1TO GZ:
      READ GE$(I): READ GE(I): NEXTI
```

Notieren Sie sich diese Tabelle zusätzlich auf ein Blatt Papier. In den DATA-Zeilen steht als erstes der Name des Gegenstandes. Es folgt eine Zahl, die angibt, in welchem Raum sich der Gegenstand befindet (Raumzahl: vergleiche Karte). Eine Routine wird ins Kursprogramm eingefügt. Sie sorgt dafür, daß bei jeder Raumbeschreibung die sich im Raum befindlichen Gegenstände aufgezählt werden. Laden Sie dazu »LISTING 14« von der beiliegenden Diskette und analysieren Sie die Basic-Zeilen. Verwenden Sie sie im Gesamtprogramm. Wenn Sie jetzt im Spielplan »herumlaufen«, können Sie problemlos feststellen, wo sich die einzelnen Gegenstände befinden.

Wenden wir uns den Objekten zu (Türen, Fenster, Schränke usw.). Zuerst muß eine entsprechende Tabelle definiert werden. Die Objektnamen werden in OB$(1) bis OB$(OZ) untergebracht. Außerdem legen wir ein Feld OO(1) bis OO(OZ) an, in dem gespeichert wird, wo sich die einzelnen Objekte befinden. Dies geschieht mit »LISTING 15« (das Sie zum Ansehen von der beiliegenden Diskette laden können):

```
52400 REM OBJEKTTABELLE-------
52410 DATA TRUHE,5
52412 DATA SCHACHT,6
52414 DATA EISENRING,6
52416 DATA TUER,2
52418 DATA TUER,5
52500 OZ=5:DIM OB$(OZ):DIM OO(OZ):FORI=1TO OZ:
      READ OB$(I): READ TO(I): NEXTI
```

Was bleibt noch zu tun? Es muß eine Routine geschrieben werden, die bei jeder Raumbeschreibung die Objekte aufzählt (analog zu den Gegenständen). Da es stilistisch schöner ist, die Objekte in der Aufzählung den Gegenständen voranzustellen, fügen wir in die programmierte Routine für die Gegenstände eine Schleife ein. Sie soll die Objekte ausgeben - falls welche im Raum vorhanden sind. Diese Aufgabe erledigt »LISTING 16«. Wenn Sie es von der beiliegenden Diskette laden und LISTen, sollten Sie sich Programmzeile 1188 näher ansehen. Es handelt sich um die Schleife für die Gegenstände. Hier ist zu beachten, daß die Checkvariable IC nicht mehr auf »0« gesetzt werden darf. Die Meldung »Nichts Besonderes« soll lediglich dann erfolgen, wenn sich weder Objekt noch Gegenstand im Raum befinden.

Damit sind wir am Ende der Programmierung unserer Spielkarte angelangt. Sie haben eine Programmiertechnik kennengelernt, die besondere Merkmale aufweist:
- geringer Bedarf an Speicherplatz
- leichte Korrekturmöglichkeit der Spielkarte
- ausbaufähig

Der vorliegende Adventure-Teil ist leicht zu modifizieren. Sie können jeden Raum mit beliebigem Text versehen, Gegenstände und Objekte auf der Spielkarte verteilen und haben die Möglichkeit, das Spiel in einem beliebigen Raum beginnen zu lassen (dazu muß der ZN-Wert in Zeile 100 verändert werden). Es lassen sich nach Wunsch neue Gegenstände und Objekte in die Karte einfügen. Wichtig ist, daß die Tabellen ab Zeile 52000 ergänzt und die Variablen GZ, OZ etc. entsprechend angepaßt werden. Der Lohn Ihrer Aufmerksamkeit und Mitarbeit bei diesem Kurs ist »LISTING 17«, das Sie von der beiliegenden Diskette laden und sich die einzelnen Programmabschnitte zur Übung noch einmal ansehen sollten. Dieses inzwischen an Umfang recht stattlich gewordene Listing bildet die Grundlage für die folgenden Kursabschnitte.

Leider ist das vorliegende Rumpf-Adventure ohne jeglichen Witz. Wir möchten uns im folgenden Abschnitt damit befassen, wie man das Action-Modul mit Routinen wie NIMM, VERLIERE, INVENTUR, OEFFNE usw. programmiert.

Endlich ist es soweit – wir bringen Leben in unser bislang recht bescheidenes Adventure. Man kann die Aktionsprogrammierung in zwei Hauptgruppen aufteilen:

1. Allgemeine Action
Diese Tätigkeiten kann der Spieler jederzeit ausführen, egal im welchem Teil der Spielkarte er sich befindet (z.B. NIMM, VERLIERE, WARTE, SINGE, INVENTUR).

2. Raumspezifische Action
Hier handelt es sich um Aktionen, die nur in einem bestimmten Raum durchgeführt werden können (z.B. Öffnen einer Tür, einer Truhe oder Drücken eines Knopfes an der Zimmerwand)

Allgemeine Action
Die Verbtabelle muß um einige Verben ergänzt werden.
OEFFNE Damit öffnen Sie Türen, Kisten usw.
SCHLIESSE Das Gegenteil von OEFFNE.
SCHAUE, UNTERSUCHE Diese Verben haben Doppelfunktion.
Ohne Objekt angewandt (»SCHAUE«) geben Sie die Raumbeschreibung wieder, in Verbindung mit einem Objekt oder Gegenstand (»UNTERSUCHE TRUHE«) erhalten Sie eine Beschreibung des Gegenstands oder Objekts.
NIMM, NEHME, HOLE Gegenstände werden vom Spieler an sich genommen.
VERLIERE, LEGE, WIRF, WERFE Die Umkehrung von NIMM.
INVENTUR Durch diesen Befehl erfährt der Spieler, welche Gegenstände er zur Zeit bei sich hat.
Ergänzen wir die Verbtabelle unseres Programms:

```
52030 DATA OEFFNE, SCHLIESSE, SCHAUE, UNTERSUCHE1,
      NIMM, NEHME1, HOLE2
52035 DATA VERLIERE,LEGE1,WIRF2,WERFE3,INVENTUR
```

Der VZ-Wert in Zeile 52100 muß in VZ=22 geändert werden.
Nach dieser Ergänzung sieht unsere Verbtabelle so aus:
Verbzahl VZ:
1 – 10 Himmelsrichtungen
11 OEFFNE
12 SCHLIESSE
13 SCHAUE,UNTERSUCHE
15 NIMM,NEHME,HOLE
18 VERLIERE,LEGE,WIRF,WERFE
22 INVENTUR

Sie erinnern sich bestimmt an den Abschnitt über die Befehlsanalyse. Dort tauchte zum ersten Mal der Begriff »Alternativverb« auf. Das ist die Erklärung, weshalb die Verben nicht fortlaufend durchnumeriert sind. Egal, ob der Spieler NIMM, NEHME oder HOLE eingibt: Die Verbzahl ist in allen Fällen VE = 15. Dasselbe gilt für die anderen Verbfamilien. Lassen Sie uns die einzelnen Routinen der Adventure-Verben in unser Programm einbauen.

Auf unserem Spielplan haben wir bereits Gegenstände verteilt (Schwert, Seil, Schlüssel, Diamant). Wir wollen unserem Spieler erlauben, diese Sachen einsammeln zu können. Das Action-Modul muß um »LISTING 18«, »LISTING 19«, »LISTING 20« und »LISTING 21« ergänzt werden. Sie finden diese Unterprogramme auf der beiliegenden Diskette. Nach dem Laden sollten Sie sich die Listings auf dem Bildschirm betrachten, um deren Programmierung zu verstehen. Um ihre Funktionsweise auszuprobieren, müssen Sie das erweiterte Übungsprogramm für unseren Adventure-Kurs, »LISTING 22«, laden und starten. Geben Sie folgende Befehle ein:

```
NIMM SCHWERT, S, N
```

NIMM-Routine
Sie haben in Raum 1 das Schwert genommen. Anschließend sind Sie nach S und N (zurück nach Raum 1) gegangen. Das Schwert wird in der Raumbeschreibung nicht mehr erwähnt. Wie funktioniert die Routine? Gehen wir von der Befehlseingabe »NIMM SCHWERT« (ohne Richtungsangaben) aus. Das Befehlsanalyse-Modul liefert an das Action-Modul folgende Werte:
VE = 15 (Verbzahl)
G1 = 1 (1. Gegenstandszahl)
Die NIMM-Routine in der Analyse der Programmzeilen:
2110 Ist die Verbzahl VE < > 15, wird die NIMM-Routine übersprungen.
2120 Befindet sich der Gegenstand GE(G1) nicht im gleichen Raum wie der Spieler, erscheint eine Fehlermeldung.
2125 Wenn der Spieler den Gegenstand bereits hat, macht uns das Programm darauf aufmerksam.
2130 Ist der Gegenstand im Raum vorhanden (GE(G1) = ZN), nimmt ihn der Spieler (GE(G1) = –1).
Daraus resultiert: Jeder Gegenstand im Besitz des Spielers erhält den Wert –1.

INVENTUR-Routine
Mit dem Befehl »INVENTUR« erfährt der Spieler jederzeit, welche Gegenstände sich in seinem Besitz befinden. Die Routine besteht im Prinzip aus einer Schleife, die die Gegenstandstabelle durchläuft und überprüft, welche Dinge den Wert **–1** haben.

VERLIER-Routine
Sie arbeitet wie die NIMM-Routine. Der gravierende Unterschied besteht in der Abfrage nach dem Gegenstand. Ist der Wert der Variablen GE(G1) anders als »–1«, erhalten Sie die Meldung »Ich habe das nicht!«. Ansonsten wird der Gegenstand im aktuellen Raum abgelegt.

SCHAU-Routine
Der Befehl SCHAU (ohne Objekt oder Gegenstand) zeigt bzw. wiederholt eine Raumbeschreibung. Ist ein Gegenstand oder ein Objekt dabei, werden diese Dinge beschrieben. Der Sprung »GOTO 1130« in Zeile 2410 aktiviert eine erneute Ausgabe der Raumbeschreibung.
Sind Sie inzwischen ein Profi im Programmieren von Adventures geworden? Zumindest sind Sie nicht mehr weit davon entfernt. Sie besitzen das nötige Wissen zum Programmieren von großen Spiellandschaften, in denen man herumlaufen und Gegenstände transportieren kann. Sie müssen noch lernen, wie man Aufgaben ins Spiel einbaut, die der Spieler lösen muß. Der nächste Abschnitt befaßt sich mit raumspezifischen Aktionen.

Action in Räumen

Eine auf den Raum zugeschnittene Aktion bereitet den größten Programmieraufwand. Und der bringt oftmals ein verheerendes Chaos in die zu Beginn gut strukturierten Listings. Dies darf uns nicht passieren.
Die bequemste Lösung ist, raumspezifische Aktionen kurzerhand an das Action-Modul zu hängen. Man kann für jeden Raum ca. 100 Zeilen reservieren (z.B. Programmzeile 2500 bis 2599 für Raum 1) und davor eine IF-THEN-Abfrage setzen. Sie müßte feststellen, ob der Raum übersprungen oder behandelt werden soll. Der Nachteil: Dieses Verfahren verzögert die Bearbeitungsgeschwindigkeit des Adventures ganz erheblich. Um diesem Problem auszuweichen, schreiben wir unsere raumspezifische Aktion direkt auf die Spielkarte.
Betrachten wir erneut ihren Aufbau und nehmen als Beispiel Raum 1. Er befindet sich in den Zeilen 10100 bis maximal 10199. Am Beginn dieses Programmteils steht eine DATA-Zeile mit den möglichen Richtungen und deren Zielorten. Die entsprechenden Zeiger werden auf diese Zeile gesetzt:

```
RESTORE 10000 + ZN*100
```

Eine READ-Schleife liest die Daten.

Die Raumbeschreibung wird mit folgender Anweisung auf dem Bildschirm ausgegeben:

```
GOSUB 10000 + ZN*100
```

Den entsprechenden Beschreibungstext finden wir in Form von PRINT-Zeilen ab Zeile 10105. Was hindert uns daran, die raumspezifische Aktion in die Programmzeilen ab 10120 bis 10199 zu legen? Das allgemeine Grundschema für einen einzelnen Raum in der Spielkarte inklusive Aktion zeigt Ihnen Bild 20. Daraus ersehen Sie, daß die raumspezifische Aktion zu Raum ZN (ZN = Zimmernummer der Karte) mit folgender Basic-Anweisung aufgerufen wird:

```
GOSUB 10000 + ZN*100 + 20
```

Dieser Programmierbefehl ist identisch mit der entsprechenden Anweisung des herkömmlichen Basic 2.0 des C 64:

```
GOSUB 10120
```

Raumspezifische Aktionen sollten am Ende des Action-Moduls aufgerufen werden. Die nötigen Programmzeilen lauten:

```
2500 REM AUFRUF RAUMSPEZIFISCHE AKTION
2510 GOSUB 10000+ZN*100+20
2600 GOTO 1000
```

Diese drei Basic-Zeilen sind als »LISTING 23« auf der beiliegenden Diskette gespeichert.

Dies ist die übersichtlichste Methode, unser Adventure mit jeder erdenklichen Art einer Aktion auszustatten. Der Vorteil liegt vor allem in der unkomplizierten Möglichkeit, eventuelle Fehler im Spielverlauf nachträglich zu beseitigen. Ein Beispiel: Wir sind in Raum 1, und irgendetwas funktioniert mit der Aktion nicht so, wie wir uns das vorgestellt haben. Wir listen die Zeilen 10100 bis 10199 und suchen gezielt nach dem Fehler.

Türen und Durchgänge programmieren

Unsere nächste Aufgabe ist, eine Tür zwischen Raum 2 und 5 zu programmieren. Bislang konnten wir Raum 5 durch die Eingabe von »S« in Raum 2 erreichen. Die Tür wird bereits in der Raumbeschreibung erwähnt, da sie in der Objekttabelle vertreten ist. Hier ist zu beachten: Jeder Durchgang muß zweimal in der Objekttabelle vertreten sein. Man sollte eine Tür, die zwei betroffene Räume voneinander trennt, von beiden Seiten aus sehen können. Vergleichen Sie dazu in »LISTING 22« die Zeilen 52416 und 52418:

```
52416 DATA TUER,2
52418 DATA TUER,5
```

Es hört sich paradox an: Der erste Schritt zur Programmierung der Tür besteht darin, den Verbindungsweg zwischen den beiden Räumen zu entfernen. Für den Durchgang in unserem Beispiel (Raum 2 und 5) müssen wir die Richtungs-DATA-Zeilen der beiden Räume ändern:

```
10202 DATA 0,0,1,0,0,0,0,6,0,0
10502 DATA 0,0,0,0,0,0,0,0,0,0
```

Damit ist die Verbindung zwischen Raum 2 und Raum 5 unterbrochen. Wir benötigen eine Variable, die den Zustand der Tür bestimmt. Drei Möglichkeiten sind denkbar:

- Die Tür ist offen. Der Variablenwert der Tür soll in diesem Fall »0« sein.
- Die Tür ist zu und kann mit »OEFFNE TUER« aufgemacht werden. Der Variablenwert muß »1« sein.
- Die Tür ist zu und kann nur mit einem Hilfsmittel (z.B. Schlüssel) geöffnet werden. Hierbei soll der Variablenwert »2« betragen.

Als Feldvariablen für Durchgänge legen wir TU(1) bis TU(X) fest. Sie sind für den Zustand aller Dinge verantwortlich, die man öffnen und schließen kann. Zunächst muß es möglich sein, die Tür ohne Schlüssel zu öffnen. Die Türvariable TU(1) erhält den Wert »1«.

Folgende Zeilen müssen in das Programmlisting aufgenommen werden:

```
52900 REM ALLGEMEINE VARIABLEN
52910 TU(1) = 1 : REM TUER 2/5
```

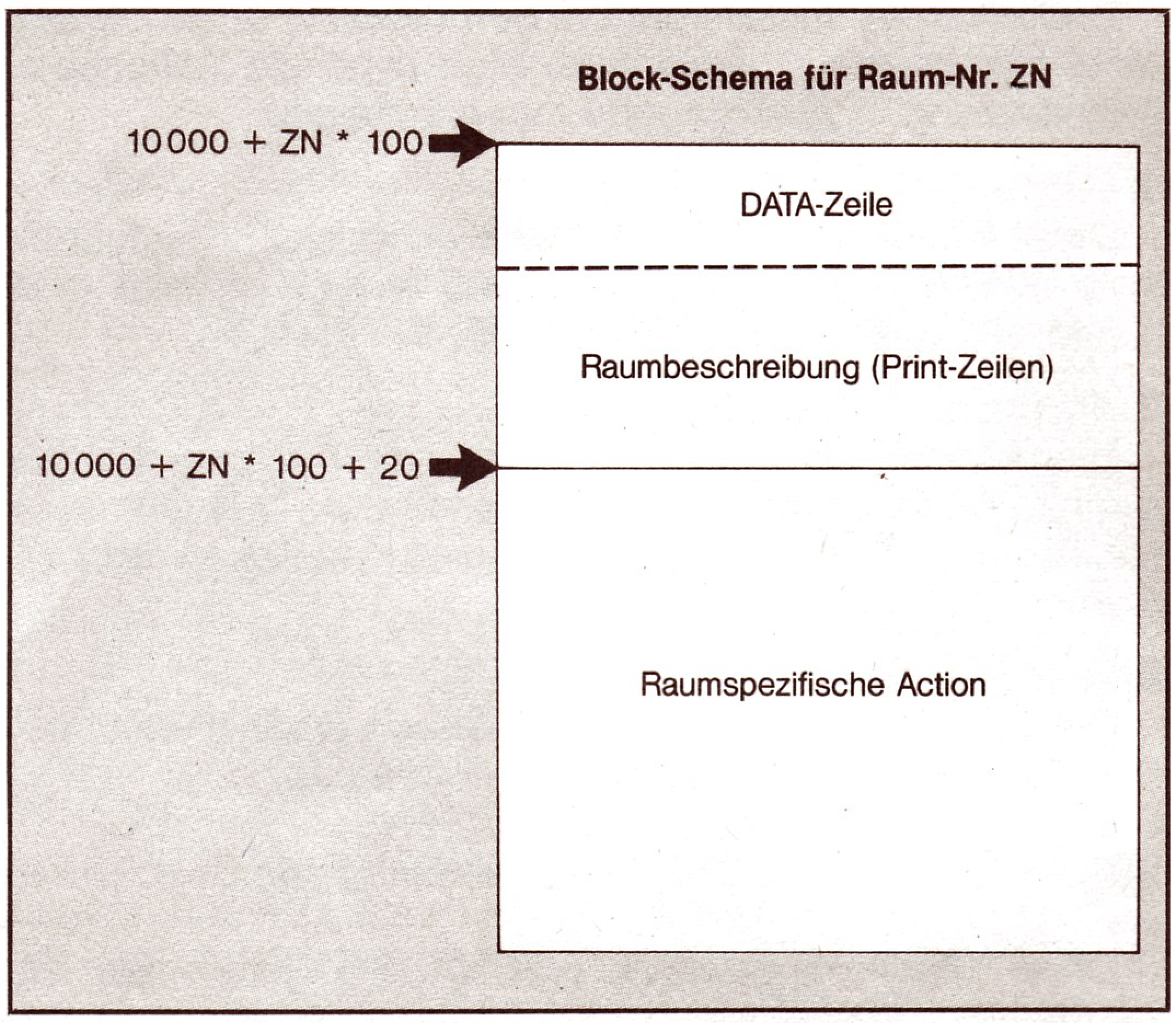


Bild 20. Programmierschema für den Raum ZN

Jetzt müssen wir die raumspezifische Aktion in den Räumen 2 und 5 zum Öffnen und Schließen der Tür programmieren:

```
10220 IF TU(1)=0 THEN RI(2)=5
10225 IF VE=11 AND OB=5 AND TU(1)=1 THEN  PRINT
      "OK.": TU(1)=0: RI(2)=5
10230 IF VE=12 AND OB=5 AND TU(1)=0 THEN PRINT
      "OK.": TU(1)=1: RI(2)=0
10250 RETURN
```

Um den einwandfreien Ablauf des Programms zu gewährleisten, wird im Action-Modul eine kleine Änderung vorgenommen:

```
1130 REM
1155 GOSUB 10000+ZN*100
```

Eine Erklärung für diese Änderung folgt später. Analysieren wir zunächst die Programmzeilen für die »Türprogrammierung« in Raum 2:

10220 Der Zustand der Tür wird überprüft. Stellt das Modul fest, daß die Tür offen ist, gilt RI(2)=5. Die Variable RI(2) ist für die Richtung SUEDEN (S) zuständig. Unter der Voraussetzung, daß die Tür offen ist, muß RI(2) auf den Wert »5« gesetzt werden. Dies gewährleistet, daß man durch die Tür in Raum 5 kommt. Der Trick der Türprogrammierung besteht darin, daß man zunächst den Verbindungsweg zwischen den Räumen entfernt. Im Falle einer offenen Tür wird der Durchgang wieder hergestellt. Das ist der Grund, warum wir das Action-Modul verändern mußten. Betritt der Spieler einen neuen Raum, geschieht folgendes:

- Der DATA-Zeiger wird mit der Anweisung RESTORE 10000+ZN*100 auf die DATA-Zeile des neuen Raums gesetzt.
- Der READ-Befehl liest die Richtungsmöglichkeiten RI(1) bis RI(10).
- Durch GOSUB 10000 +100*ZN ruft das Programm die Raumbeschreibung auf. Ist die Tür offen, wird der Verbindungsweg wieder hergestellt.

Die Änderung des Action-Moduls bestand in der Vertauschung der Arbeitsschritte 2 und 3. Nur in dieser Anordnung funktioniert das Programm einwandfrei.

10225 Hier wird geprüft, ob die Befehlseingabe des Spielers »OEFFNE TUER« lautet. Das Modul sieht im Variablenspeicher nach, ob VE=11 und OB=5 ist. Außerdem wird überprüft, ob die Tür tatsächlich ge-

schlossen ist. Treffen alle Voraussetzungen zu, wird TU(1) auf den Wert »1« (Tür ist offen) gesetzt. Der Verbindungsweg zu Raum 5 ist hergestellt: RI(2)=5.

10230 Es gilt dasselbe wie für Zeile 10225, allerdings wird die Tür geschlossen.

Sie fragen mich, wie die Abfrage nach dem Wort »TUER« im Eingabesatz des Spielers erfolgt? Hier ist die Antwort:

```
IF OB=5 THEN...
```

Wieso muß die Variable OB den Wert »5« (und nicht »4«) besitzen? Werfen wir einen Blick auf die Objekttabelle:

```
OB$(1)="TRUHE"       OO(1)=5
OB$(2)="SCHACHT"     OO(2)=6
OB$(3)="EISENRING"   OO(3)=6
OB$(4)="TUER"        OO(4)=2
OB$(5)="TUER"        OO(5)=5
```

Wie Sie wissen, stammt der OB-Wert aus dem Befehlsanalyse-Modul. Dieses durchläuft mit einer Schleife die Objekttabelle:

```
FOR I=1 TO OZ : IFBE$(WZ) = OB$(I) THEN OB=I
```

Damit ist klar, warum der erhaltene OB-Wert »5« und nicht »4« ist. Wir müssen darauf achten, daß für Objekte mit gleichem Namen immer der OB-Wert des Objekts ausgegeben wird, welches in der Tabelle zuletzt gefunden wird (es besitzt den höchsten Wert).

Raum 5 muß jetzt mit der gleichen spezifischen Aktion versehen werden, die in Raum 2 enthalten ist (Öffnen und Schließen der Tür). Im Programm erledigen dies die Zeilen von »LISTING 24« auf der beiliegenden Diskette, die Sie sich zu Studienzwecken ansehen sollten.

Im Prinzip ist die Programmierung der Türen und Durchgänge abgeschlossen. Trotzdem ist es erstrebenswert, dem Spieler mit der Anweisung »GEH TUER« den Durchlaß zu ermöglichen. Bislang muß man nach dem Öffnen der Tür den Befehl »SCHAU« eingeben, um zu erfahren, welche weitere Richtung sich durch das Öffnen ergeben hat. Die Verbtabelle wird mit dem Verb »GEH« erweitert:

```
52030 DATA GEHE,BETRETE1
```

Der VZ-Wert in Zeile 52100 muß auf »24« erhöht werden.

Die raumspezifische Aktion in Raum 2 und 5 sollte erweitert werden:

```
10240 IF VE=23 AND OB=5 THEN VE=2
10540 IF VE=23 AND OB=5 THEN VE=1
```

Wir müssen das Action-Modul ergänzen, damit es diese Routinen erkennt:

```
1110 IFRI(VE) = 0 THEN PRINT "KEIN WEG IN DIESE
     RICHTUNG !": VE = 0:GOTO1200
1130 VE=0
2520 IFVE>0 AND VE<11 THEN 1100
```

Falls Ihnen unklar ist, weshalb diese Änderungen unbedingt nötig sind, dann spielen Sie die Türszene zuerst vorher und anschließend nach der Änderung des Action-Moduls durch.

Momentan kann die Tür problemlos geöffnet werden. Ab sofort wollen wir das Öffnen und Schließen von einem Schlüssel abhängig machen. Die Tür läßt sich nur öffnen, wenn der Spieler im Besitz des Schlüssels ist. Viele englische Abenteuerspiele verlangen als Befehlseingabe zum Öffnen einer verschlossenen Tür mit einem Schlüssel etwa folgende Eingaben:

```
ENTRIEGLE TUER
OEFFNE TUER
GEH TUER
```

In der deutschen Sprache ist es allgemein nicht üblich, »Entriegle Tür« zu sagen. Wir wollen diesen Ausdruck weglassen und folgenden Ablauf vereinbaren:

- Will der Spieler die Tür mit »OEFFNE TUER« ohne passenden Schlüssel aufmachen, erhält er die Meldung »ICH HABE KEINEN PASSENDEN SCHLUESSEL«.
- Hat er einen Schlüssel bei sich und gibt als Befehl »OEFFNE TUER« ein, erhält er die Meldung »OK.«.

Es ist pure Haarspalterei, vom Spieler Eingaben wie z.B. »OEFFNE DIE TUER MIT DEM SCHLUESSEL« bzw. »ENTRIEGLE DIE TUER MIT DEM SCHLUESSEL« und anschließend »OEFFNE TUER« zu verlangen. Die Frage nach dem Schlüssel läßt sich unkompliziert programmieren: Wir müssen vor die »OEFFNE TUER«-Routine eine Abfrage einbauen, die feststellt, ob GE(3) den Wert »-1« hat (ist der Schlüssel im Besitz des Spielers?). Die notwendigen Ergänzungszeilen finden Sie in »LISTING 25« auf der beiliegenden Diskette:

```
10224 IF VE=11 AND OB=5 AND GE(3)<>-1 THEN PRINT
      "ICH HABE KEINEN SCHLUESSEL.":GOTO 10230
10524 IF VE=11 AND OB=5 AND GE(3)<>-1 THEN PRINT
      "ICH HABE KEINEN SCHLUESSEL.":GOTO 10230
```

Wenn Sie das Programm ergänzt haben, machen Sie ein Testspiel und probieren das Öffnen der Tür mit dem Schlüssel aus. Spielen Sie folgende Variante durch:

1. Schlüssel holen
2. Tür in Raum 2 öffnen.
3. Schlüssel verlieren.
4. In Raum 5 gehen.
5. Tür schließen.

Sie sind in Raum 5 und können diesen nicht mehr verlassen, da das Türschloß zugeschnappt ist. Der Schlüssel liegt in Raum 2. Damit kann man den Spieler in eine Falle laufen lassen. Es ist eine Tatsache, daß kaum ein Spieler eine Tür wieder schließt, nachdem er sie geöffnet hat. Die Programmierung eines derartigen Effekts (sie ist in der bisherigen Türen-Logik bereits enthalten) soll den Spieler verblüffen.

Geheimnisse einer Truhe

Befinden Sie sich in Raum 5, werden Sie feststellen, daß dort eine Truhe steht. Transportieren (nehmen) kann man diese nicht. Ich habe beschlossen, daß Objekte vom Spieler nicht fortgetragen werden können, da sie zu schwer sind. Dazu muß eine weitere Fehlermeldung in das Adventure eingebaut werden. Versucht der Spieler, ein Objekt zu nehmen (Tür, Truhe usw.), erhält er die Antwort »DAS GEHT UEBER MEINE KRAEFTE!«. Dieser Hinweis im Spiel läßt sich in die NIMM-Routine einbauen:

```
2115 IF OB<>0 THEN PRINT "DAS GEHT UEBER MEINE
     KRAEFTE!" : GOTO2200
```

Es spricht nichts dagegen, daß in der Truhe Gegenstände liegen. Wir legten für unser Programm fest: Ist die zuständige Variable GE(X) größer als »0«, liegt der Gegenstand im Raum. Hat GE(X) den Wert »-1«, befindet sich das Ding im Besitz des Spielers. Wir ergänzen das Abenteuerspiel mit folgender Bedingungsabfrage: Besitzt GE(X) den Wert »-2«, liegt der Gegenstand in der **Truhe**. Wir wollen unsere Gegenstandstabelle dergestalt ändern, daß das Schwert nicht länger in Raum 1 liegt. Es soll sich in der Truhe befinden:

```
52210 DATA SCHWERT,-2
```

Ab sofort ist das Schwert in der Truhe versteckt. Wie stellen wir es an, im Verlauf des Spiels das Schwert wieder an uns zu bringen?

Ihre Antwort ist völlig richtig: Wir müssen die Truhe öffnen. Dazu ist es erforderlich, in die raumspezifische Aktion von Raum 5 eine weitere Programmroutine einzufügen. Sie soll das OEFFNEN der TRUHE erlauben. Prinzipiell kann eine Truhe, ein Schrank oder eine Kiste wie eine Tür betrachtet werden. Hierbei sind ebenfalls drei Zustandsformen möglich: offen, geschlossen, verriegelt.

Wir benötigen eine neue Variable, die Auskunft über den jeweiligen Zustand der Truhe gibt. Nennen wir diese Variable TU(2). Gehen wir davon aus, daß der Ausgangszustand der Truhe »1« ist (Truhe ist geschlossen und kann mit dem Befehl »OEFFNE TRUHE« ohne zusätzliches Hilfsmittel wie Schlüssel usw. geöffnet werden).

Die Zeile im Programmlisting muß folgendermaßen lauten:

```
52920 TU(2)=1:REM TRUHE
```

Die OEFFNE-Routine für Raum 5:

```
10545 IF VE=11 AND OB=1 AND TU(2)=1 THEN PRINT
      "OK.":TU(2)=0
```

Was man öffnen kann, läßt sich wieder schließen:

```
10546 IF VE=12 AND OB=1 AND TU(2)=0 THEN PRINT
      "OK.":TU(2)=1
```

Sie erkennen daran, daß diese zusätzlichen Abfrageroutinen problemlos zu programmieren sind. Es macht sich bezahlt, daß wir zu Beginn viel Denkarbeit in die Programmierung des Befehlsanalyse-Moduls gesteckt haben. Wir sind dadurch in der Lage, beliebig weitere Routinen anzufügen. Dazu muß lediglich der Wortschatz entsprechend erweitert, in der Routine selbst VERBZAHL, OBJEKTZAHL usw. abgefragt und die TUER-Variablen etc. geändert werden.

Wir sind jetzt in der Lage, die Truhe zu öffnen und zu schließen. Vom Schwert sehen wir aber keine Spur. Dazu müssen wir die Raumbeschreibungsroutine erweitern. Die Gegenstände in der Truhe sollten künftig in der Raumbeschreibung erwähnt werden (vorausgesetzt, die Truhe ist geöffnet). Eine einzige Abfrage genügt:

```
1189 IF GE(I)=-2 AND TU(2)=0 AND ZN=5 THEN PRINT
     GE$(I);",";:IC=1
```

Die Abfrage steht innerhalb der Schleife, in der geprüft wird, ob sich im betreffenden Raum ein Gegenstand befindet. Die »Truhenabfrage« muß berücksichtigen, daß der Gegenstand nur dann in der Raumbeschreibung vorkommen darf,

- wenn der Spieler sich in Raum 5 befindet (dort steht die Truhe),
- wenn die Truhe offen ist,
- wenn darin Gegenstände liegen (GE-Wert = -2).

Gehen Sie in Raum 5 und geben zuerst »OEFFNE TRUHE« und anschließend den Befehl »SCHAU« ein. Das Schwert wird sichtbar. Schließen Sie die Truhe wieder, verschwindet das Schwert aus der Raumbeschreibung.

Angenommen, das Schwert ist zu sehen. Sollten Sie versuchen, sich das Schwert mit »NIMM SCHWERT« anzueignen, werden Sie über die Fehlermeldung »ICH SEHE DIESEN GEGENSTAND HIER NICHT!« enttäuscht sein. Unser Programm fragt nämlich nur, ob der Gegenstandswert der gewünschten Sache mit der Raumnummer ZN identisch ist (GE(G1) = ZN).

Eine neue Abfrage ist nötig. Sie soll erlauben, Gegenstände aus der Truhe zu nehmen, wenn diese offen ist. Voraussetzung: Der Spieler muß sich im selben Raum befinden, in dem die Truhe steht. Die entsprechende Programmzeile sollte ebenfalls in der raumspezifischen Aktion von Raum 5 stehen:

```
10548 IF VE=15 AND GE(G1)=-2 AND TU(2)=0 THEN PRINT
      "OK.":GE(G1)=-1
```

Jetzt können Sie Gegenstände aus der Truhe nehmen. Ein Problem bleibt: Die Meldung »ICH SEHE DIESEN GEGENSTAND HIER NICHT!« erscheint noch immer. Dieser Widerspruch muß beseitigt werden. Vor die Programmzeile, die diese Fehlermeldung produziert, wird eine Abfrage gesetzt:

```
2119 IF GE(G1)=-2 AND ZN=5 AND TU(2)=0 THEN2125
```

Die Programmierung rund um das Objekt »Truhe« ist damit noch nicht beendet.

Um Gegenstände in die Truhe »LEGEN« zu können, müssen wir eine objektbezogene VERLIER-Routine programmieren. Laut unserer Worttabelle besteht die VERLIER-Wortfamilie aus folgenden Mitgliedern: VERLIERE, LEGE, WIRF, WERFE.

Wir nehmen an, der Spieler steht in Raum 5 und hat die Truhe geöffnet. Gibt er den Befehl »VERLIER Gegenstand« ein, nimmt dieses Ding den Wert der Raumnummer ZN an (GE(G1)=ZN). Es befindet sich ab sofort in Raum 5. Wir müssen eine Routine programmieren, die dem Gegenstand bei einer Eingabe wie z.B. »LEGE (GEGENSTAND) IN TRUHE« den Wert »-2« verleiht. Dazu eignen sich erneut die Programmzeilen der raumspezifischen Aktion von Raum 5:

```
10550 IF VE=18 AND OB=1 AND TU(2)=0 AND GE(G1)=-1
      THEN PRINT "OK.":GE(G1)=-2
10590 RETURN
```

Zusätzlich muß eine Abfrage in die VERLIER-Routine eingebaut werden.

```
2301 IF OB<>0 THEN 2400
```

Dazu muß die Truhe zunächst als Raum definiert werden.

```
10700 REM IN DER TRUHE ----
10702 DATA 0,0,0,0,0,0,0,0,0,0
10705 PRINT "IN DER TRUHE."
10720 RETURN
```

Um in die Truhe zu kommen, muß der Spieler in Raum 5

Ein Gespenst geht um...

den Befehl »GEH TRUHE« eingeben. Vorher muß er sie öffnen. Wir ergänzen die raumspezifische Aktion in Raum 5:

```
10560 IF VE=23 AND OB=1 AND TU(2)=0 THEN RI(1)=
      7:VE=1
```

In diese Programmzeile ist der Trick des GEH-Befehls integriert: In die Richtungsvariable RI(1) wird der Wert des Zielraums geschrieben (TRUHE = 7) und die Verbzahl VE auf die Zahl »1« gesetzt. Dieser Vorgang aktiviert das GEHEN.

Sie befinden sich jetzt in der Truhe. Enttäuscht werden Sie feststellen, daß das Schwert nirgends zu sehen ist. Dies liegt daran, daß wir festgelegt haben: Jeder Gegenstand **in** der Truhe erhält den Wert »-2«. Da die Truhe selbst zu einem Raum geworden ist, muß allen Gegenständen darin der Wert »7« (Raumnummer der Truhe) zugewiesen werden. Dies funktioniert problemlos: Wandeln Sie alle Werte »-2« in den entsprechenden Listingzeilen in die Zahl »7« um.

Irgendwann möchten wir aus der Truhe wieder herausklettern. Dies erfordert erneut eine Erweiterung der raumspezifischen Aktion von Raum 7. Dem Wortschatz wird das Verb VERLASSE hinzugefügt und die Aktion programmiert:

```
52045 DATA VERLASSE
52100 VZ=25 usw. (VZ-Wert anpassen!)
10720 IF VE=25 AND OB=1 AND TU(2)=0 THEN RI(1)=
      5:VE=1
```

Es handelt sich im Prinzip um die Umkehrung der Abfrage für das Klettern in die Truhe. Fällt Ihnen dabei etwas auf? In dieser Programmzeile wird überprüft, ob die Truhe offen ist.

Programmierung von Personen

Bereits zu Beginn unseres Kurses habe ich erwähnt, daß Adventures mit mehreren Personen (Spielfiguren, die vom Programm gesteuert werden – nicht der Spieler selbst) besonders reizvoll sind. Fälschlicherweise hält man die Programmierung solcher Charaktere für außerordentlich schwierig. Tatsächlich ist es relativ einfach, Nichtspielercharaktere in einem Abenteuerspiel agieren zu lassen.

Das erste Problem besteht darin, einen Weg zu finden, wie eine zusätzliche Person (in unserem Falle ein Gespenst) in der Spielkarte herumlaufen kann. Eine Möglichkeit ist, das Gespenst zufallsgesteuert (RND-Funktion) von Raum zu Raum irren zu lassen. Diese Lösung erweist sich auf die Dauer als zu simpel. Der Geist könnte sich in einer Sackgasse verfangen und dort bis zum Ausschalten des Computers herumlaufen. In so einem Fall trifft der Spieler äußerst selten auf das Gespenst. Wir wollen das Gegenteil erreichen: Der Poltergeist soll dem Spieler möglichst oft in die Quere kommen, um ihn nervös zu machen. Eine weitere, nicht weniger primitive Lösung wäre, das Gespenst gar nicht herumlaufen zu las-

sen. Per Zufall gesteuert könnte es plötzlich im Raum des Spielers auftauchen.

Die folgende Methode hat sich in meinen bisher programmierten Adventures bewährt. Zunächst muß das Gespenst in die Personentabelle aufgenommen werden:

```
52600 REM PERSONENTABELLE ----
52610 DATA GESPENST
52700 PZ=1:DIMPE$(PZ):FORI=1TOPZ:READPE$(I):NEXT
```

Als nächster Schritt wird eine exakte Route für die Wanderung des Geistes im Spiel geplant. Für unser Adventure könnte dies ein möglicher Idealweg sein: 1-3-4-3-1-2-5-2-6-2-1. Die Zahlen stellen die einzelnen Raumnummern dar. Das Gespenst beginnt in Raum 1, geht nach Raum 3 usw. Wichtig ist, daß der letzte Raum in der Kette wieder mit dem ersten Raum identisch ist. Schließlich wollen wir auch einem Geist nicht erlauben, durch Wände gehen zu können und damit von einem Raum in den anderen zu kommen. Das Gespenst läuft ständig die gleiche Route ab. Dies hört sich langweilig an. Ich darf Ihnen versichern: Es wird dem Spieler nicht auffallen. Zugegeben, bei unserem Mini-Übungs-Adventure ist es nicht schwer, die Marschroute des Geistes zu durchschauen. Bei Spielen mit 100 und mehr Räumen jedoch besteht keine Chance, die Route eines Nichtspielercharakters herauszufinden – außer, man hat das Spiel selbst programmiert.

Ein weiterer Vorteil der Routenprogrammierung liegt darin, daß man die Reichweite für die einzelnen Figuren begrenzen kann. Die Marschroute wird durch das Feld PE(1) bis PE(11) festgelegt.

```
52920 DATA1,3,4,3,1,2,5,2,6,2,1
52935 DIMPE(11):FORI=1TO11:READPE(I):NEXT:MO=1
```

Die Steuerung des Gespenstes soll innerhalb des Action-Moduls ab Zeile 3000 beginnen:

```
3000 REM STEUERUNG DES GESPENSTES
3001 PRINT "GESPENST=";PE(MO)
3010 IF MO=0 THENRETURN
3020 MO=MO+1:IF MO=12 THENMO=1
3025 IFPE(MO)<>ZNTHEN3100
3100 RETURN
```

Aufgerufen wird dieses Unterprogramm, bevor die raumspezifische Aktion beginnt:

```
2505 GOSUB 3000 : REM GESPENST
```

Hier die Erläuterung der »Gespenster«-Routine:

Zeile 3001 dient dazu, den Geist zu verfolgen. Sie kann jederzeit ersatzlos gestrichen werden. Die Variable MO zählt von »1« bis »11« und beginnt wieder von vorne: PE(MO) ist der Raum, in dem sich das Gespenst gerade befindet. Hat MO den Wert »0«, ist das Gespenst nicht aktiv. Das Programm kehrt zum Hauptmodul zurück (Zeile 3010). Nach jedem Spielzug des Spielers wird die Variable MO um »1« erhöht – das Gespenst geht um... Um den Spieler die Anwesenheit des Geistes spüren zu lassen, müssen wir folgende Zeile einfügen:

```
3030 PRINT"EIN RIESIGES GESPENST ERSCHEINT!"
```

Unser Gespenst ist bislang ausgesprochen harmlos. Folgende Aufgabe werden wir ihm zuteilen:

Betritt es einen Raum, in dem ein Gegenstand liegt, soll es diesen an sich nehmen. Der Geist darf nur einen einzigen Gegenstand transportieren und ihn gegen einen anderen austauschen können. Die Ergänzung unseres Programms lautet:

```
3100 IC=0:FORI=1TOGZ:IFGE(I)=PE(MO)THENIC=I
3105 NEXT
3110 IF IC=0 THEN3150
3120 GE(GF)=PE(MO):GF=IC:GE(IC)=0
3150 RETURN
```

Innerhalb einer Schleife wird überprüft, ob sich in dem Raum mit dem Gespenst ein Gegenstand befindet. Trifft dies zu, legt das Gespenst das Ding, das es momentan bei sich trägt, in diesem Raum ab (GE(GF)=PE(MO)) und nimmt das neue mit (GF=IC). Ein Gegenstand im Besitz des Gespenstes ergibt den Variablenwert GE(IC)=0. Die Variable GF dient als Zwischenspeicher für genommene Gegenstände.

Wir können es nicht lassen. Ein weiterer Effekt wird ins Programm aufgenommen:

```
3150 IF PE(MO)=5 AND ZN=7 AND TU(2)=0 THEN TU(2)=1:
     PRINT "JEMAND SCHLIESST DIE TRUHE"
3160 RETURN
```

Betritt der Geist Raum 5, und der Spieler befindet sich zu diesem Zeitpunkt in der Truhe, wird diese vom Gespenst zugeklappt. Der Spieler ist gefangen!

Als Ergebnis unserer Arbeit liegt »LISTING 26« als fertiges Abenteuerspiel auf der beiliegenden Diskette vor.

Tips und Hinweise

Es empfiehlt sich als Farbe für Hintergrund und Rahmen »Schwarz« zu wählen. Als Schriftfarbe eignet sich Hellgrau, Weiß oder Grün.

Falls Sie Ihr Abenteuerspiel mit **Grafik** ausstatten möchten, sollten Sie dies erst dann in Angriff nehmen, wenn die Module und das gesamte Adventure fertig programmiert sind. Für die Größe eines Grafikbildes empfehle ich eine maximale Ausdehnung von 10 mal 20 Zeichen (Textmodus/Blockgrafik). Ganzseitige Bildschirme oder Hires-Bilder verbrauchen zuviel Platz, wenn sie zusammen mit dem Spielprogramm im Speicher des C 64 stehen. Eine unbefriedigende Alternative ist ein zeitaufwendiges Nachladen solcher Grafik-Files von Diskette.

Utility »Grafik-Designer«

Hierbei handelt es sich um einen Maskengenerator, der es Ihnen ermöglicht, problemlos Blockgrafik-Bilder zur Ausstattung Ihrer Adventures zu programmieren.

Laden Sie das Basic-Programm mit

```
LOAD "GRAFIK-DESIGNER",8
```

von der beiliegenden Diskette. Starten Sie es noch **nicht!**

Löschen Sie mit der Taste <CLR/HOME> den Bildschirm.

Mit jeder gewünschten Taste (inkl. Farben und Revers-Modus) können Sie die gewünschte Grafik auf den Bildschirm »malen«. Sind Sie am rechten Bildschirmrand angekommen, vermeiden Sie es, die RETURN-Taste zu drücken. Benutzen Sie ausschließlich die Cursortasten, um sich innerhalb des »Gemäldes« auf dem Bildschirm zu bewegen. Ist das Bild fertig, wird es an jeder Ecke (oben links und rechts, unten links und rechts) mit einem At-Sign-Zeichen (Klammeraffe) versehen. Die vier Markierungen müssen die Eckpunkte eines Rechtecks begrenzen, in dem das Bild steht.

Jetzt kann der Maskengenerator mit

```
RUN 60000-
```

gestartet werden. Von der Größe des Bildes hängt die Bearbeitungszeit ab. Nach kurzer Zeit erscheint die Frage »AB WELCHER ZEILE?« auf dem Bildschirm. Denken Sie an unser Modulsystem und wählen Sie Zeilennummern, die den Ablauf Ihres Adventure-Programms nicht stören. Wir empfehlen: ab 40000 bis maximal 49999. Ist die Eingabe erledigt, listet der »Grafik-Designer« Ihre Grafik, in PRINT-Zeilen umgewandelt, auf dem Bildschirm. Setzen Sie den Cursor auf den Anfang der ersten Zeilennummer und übernehmen Sie diese und die folgenden mit der RETURN-Taste. Zum Entwerfen und Generieren weiterer Grafiken löschen Sie erneut den Bildschirm und verfahren wie vorher beschrieben. Erzeugte Bildmodule lassen sich wie jedes andere Basic-Programm abspeichern und mit einem MERGE-Utility in Ihr Adventure einfügen. Vergessen Sie nicht, vorher die Basic-Zeilen des Maskengenerators ab Zeile 60000 zu löschen.

Ich hoffe, daß es mir mit diesem Kurs für Einsteiger gelungen ist, Sie für Abenteuerspiele und deren Programmierung zu interessieren. Dazu wünsche ich Ihnen eine Menge guter Ideen und viel Spaß. (Michael Nickles/bl)

OHNE UMWEG ZUM ZIEL

Es bringt nichts, den C64 samt Monitor aus dem Fenster zu werfen. Damit lösen Sie kein noch so verzwicktes Abenteuerspiel. Atmen Sie tief durch und lesen Sie lieber die folgenden Seiten.

Adventures zu lösen, ist oft nicht einfach. Gerade das macht den Reiz dieser Spiele aus. Viele, die vorher noch nie mit Abenteuerspielen in Berührung kamen, haben verzweifelt aufgegeben. Wir möchten Ihnen bei den ersten Schritten helfen.

Die Tips, die Sie hier erhalten, gelten nur für Text- oder Grafik-Adventures, die Eingaben über die Tastatur verlangen. Für sog. Joystick-Abenteuerspiele bzw. Rollenspiele gelten andere Regeln.

Lösungsbücher – ja oder nein?

Wer sich ein Abenteuerspiel gekauft hat und damit absolut nicht zurechtkommt, sollte ganz einfach Freunde mitspielen lassen, bevor er zu Lösungsbüchern greift. Außerdem macht das Ganze mehr Spaß. Wer dann schließlich doch zur Hilfsliteratur greift, muß zwischen Lösungsbüchern und sog. »Hint Books« unterscheiden. »Hints« sind Tips, aber keine Lösungen. Die Hersteller verschiedener Adventures bieten oftmals zu ihren Spielen Lösungshinweise an. Buch- und Zeitschriftenverlage haben in der Regel ihre Adventure-Hacker, die ein Adventure »durchackern« und Lösungen bzw. »Hint Books« anbieten.

Manche Softwarehäuser legen ihren Spielen verschlüsselte Lösungshinweise bei, die man – »schummeln« ausgeschlossen – erst mit Hilfe einer Tabelle decodieren muß.

Wortschatz ergründen

Für verschiedene kommerzielle Abenteuerspiele kann man »Hint Books« bestellen, in denen Lösungshinweise mit Geheimtinte abgedruckt sind. Sie werden nach Überstreichen mit einem mitgeliefertem Spezialfilzstift sichtbar. Dadurch ist gewährleistet, daß man jeweils nur die Informationen erhält, die man tatsächlich haben möchte.

Wer von den Herstellerfirmen bestimmter Adventures keine Lösungshinweise erhalten kann, der sollte sich sowohl in deutschen als auch amerikanischen Fachzeitschriften umsehen. Oft sind dort Lösungen und Tips zu finden.

Adventure-Spiele haben eines gemeinsam: Alle verstehen ihren Wortschatz mit Hilfe eines »Parsers«. Das ist die Bezeichnung der Sprachinterpretations-Routine, die überprüft, wie der eingegebene Satz aufgebaut ist (Unterscheidung von Verben, Substantiven und Präpositionen). Diese Wörter müssen in einer programmierten Wortschatzliste enthalten sein.

Bei sehr guten Adventures ist fast alles möglich – tippen Sie ein, was Sie denken. Wenn Ihr benutztes Wort nicht im z.B. 800 Worte umfassenden Wortschatz enthalten ist, probieren Sie ein Synonym. Meist führt das zum Erfolg.

Einigen Abenteuerspielen liegen Wortschatzlisten bei. Viele der besten Adventure-Programmierer hassen das stupide Ein-/Ausgabe-Spielchen nach dem Motto: »Guess what word the parser wants to have« (Gib nur die Wörter ein, die der Parser dir vorschreibt).

Bei anderen Adventures gilt: Grundsätzlich sollte man erst einmal alles ausprobieren und eventuell in einem Englisch-Wörterbuch nachsehen. Wenn man hier an die Grenzen gestoßen ist, kann man mit Hilfe eines guten Maschinensprachemonitors das Hex-Listing eines Programms durchforschen und entsprechende Wissenslücken ausfüllen. Am idealsten eignet sich dazu ein Diskettenmonitor, da manche Adventures aufgrund veränderter Vektoren im Betriebssystem des C64 kein Auflisten des Hexdumps zulassen. Manche Abenteuerspiele sind als Schutz vor derart »unfairem« Verhalten sehr geschickt codiert. In diesem Fall hilft nur noch ein ausgezeichnetes Vokabelgedächtnis, um sich die gängigen Befehle des Adventures zu merken.

Vor dem Spielbeginn sollten Sie immer in der Anleitung nachsehen, ob das Adventure ganze Sätze oder nur Zwei-Wort- bzw. Ein-Wort-Kommandos annimmt.

Adventure-Räume genau untersuchen

Steht man in einem Raum, erfährt man in vielen Adventures nur das, was auf den ersten Blick zu sehen ist. Geben Sie in diesem Fall unbedingt den Befehl LOOK (SIEH, SCHAUE, UMSEHEN) ein. Häufig erfahren Sie erst dann, was es bei genauerem Hinsehen noch zu entdecken gibt. Gegenstände, die herumliegen, irgendwelche Öffnungen, Türen oder Spalten sollten Sie auf jeden Fall genauer untersuchen. Das funktioniert meist mit Hilfe der Befehle EXAMINE OBJECT bzw. INSPECT OBJECT (UNTERSUCHE...). Um etwas zu durchsuchen, sollte man SEARCH IN »Gegenstand«, SEARCH AT »Gegenstand« oder, je nach Spiele-Parser, SEARCH »Gegenstand« eintippen.

Bei einigen Abenteuern erlaubt es die Zeit nicht, alle in einem Raum vorhandenen Dinge zu untersuchen. Unser Tip: Machen Sie es trotzdem. Im Lauf der Zeit bekommen Sie ein Gespür dafür, welche Gegenstände unnütz sind, und welche Sie gebrauchen können. Sie haben die Möglichkeit, nach Ablauf der Zeit wieder neu zu starten und jetzt das Untersuchen abzukürzen – Sie wissen bereits aus der vorherigen Spielrunde, wie die Gegenstände aussehen, und was sich dahinter verbirgt.

Objekte und Gegenstände notieren

Nehmen Sie nur die Dinge mit, von denen Sie bestimmt glauben, Sie würden sie brauchen. Benötigen Sie allerdings später einen Gegenstand, den Sie vorher nicht mitgenommen haben, und Sie erinnern sich nicht mehr daran, wo er lag, ist das nicht gut. Sie sollten sich deshalb alle Gegenstände aufschreiben, die Ihnen irgendwo auffallen. Auch in den Orts- bzw. Raumbeschreibungen sind oft nützliche Hinweise versteckt. Wenn Sie meinen, daß es wichtig ist: ebenfalls aufschreiben. Die Umgebung sollte außerdem ständig kartographiert werden. Darüber jedoch später mehr.

Wichtig: Aktuellen Spielstand speichern

Vor kritischen Stellen ist es ratsam, das Spiel zu speichern. Die meisten Adventures benutzen dazu den Befehl SAVE oder SAVE GAME. Manchmal kommt man mit QUIT in ein

Menü, von dem aus der aktuelle Spielstand auf Diskette zu sichern ist. Sie sollten sich überhaupt zur Gewohnheit machen, den bisherigen Spielverlauf regelmäßig in gewissen Abständen zu speichern (je nach Umfang und Schwierigkeitsgrad etwa nach einem Fünftel bis einem Zehntel des Abenteuers). Bei extrem schweren Spielen am besten nach jedem Spielzug.

Adventures, die lügen können

In verschiedenen Abenteuerspielen sind ziemlich heimtückische Fallen eingebaut. Probieren Sie alles, was Ihnen möglich erscheint, auch wenn es noch so unlogisch oder ungewöhnlich ist. Beispiel: Im 1985 entstandenen Adventure »The Hitchhikers Guide to the Galaxy« heißt es an einer Stelle, man könne nicht zum Ausgang hinaus. Nach dem dritten Versuch zeigt sich aber, daß es doch geht: Das Spiel (bzw. der Programmierer) gibt zu, daß es **lügt**! Wenn Sie der festen Überzeugung sind, Ihre Idee sei die einzig richtige, dann versuchen Sie es getrost mehrmals.

Bei zeitabhängigen Abenteuern sollten Sie nicht jedesmal ohne zu überlegen lostippen, um den Computer zu einer Aktion zu zwingen. Warten Sie ruhig ab, was passiert (WAIT, WARTE). In vielen Fällen hilft Ihnen das weiter.

Eingaben abkürzen

Obwohl das Abenteuerspiel, das Sie gerade im Computer haben, ganze Sätze versteht, können Sie durch Abkürzungen eine Menge Zeit ersparen. Tippen Sie z.B. als Kurzbezeichnungen für die Himmelsrichtungen immer nur S, W, N, SW, SO, NW, NO. Arbeiten Sie mit einem Ganz-Satz-Adventure

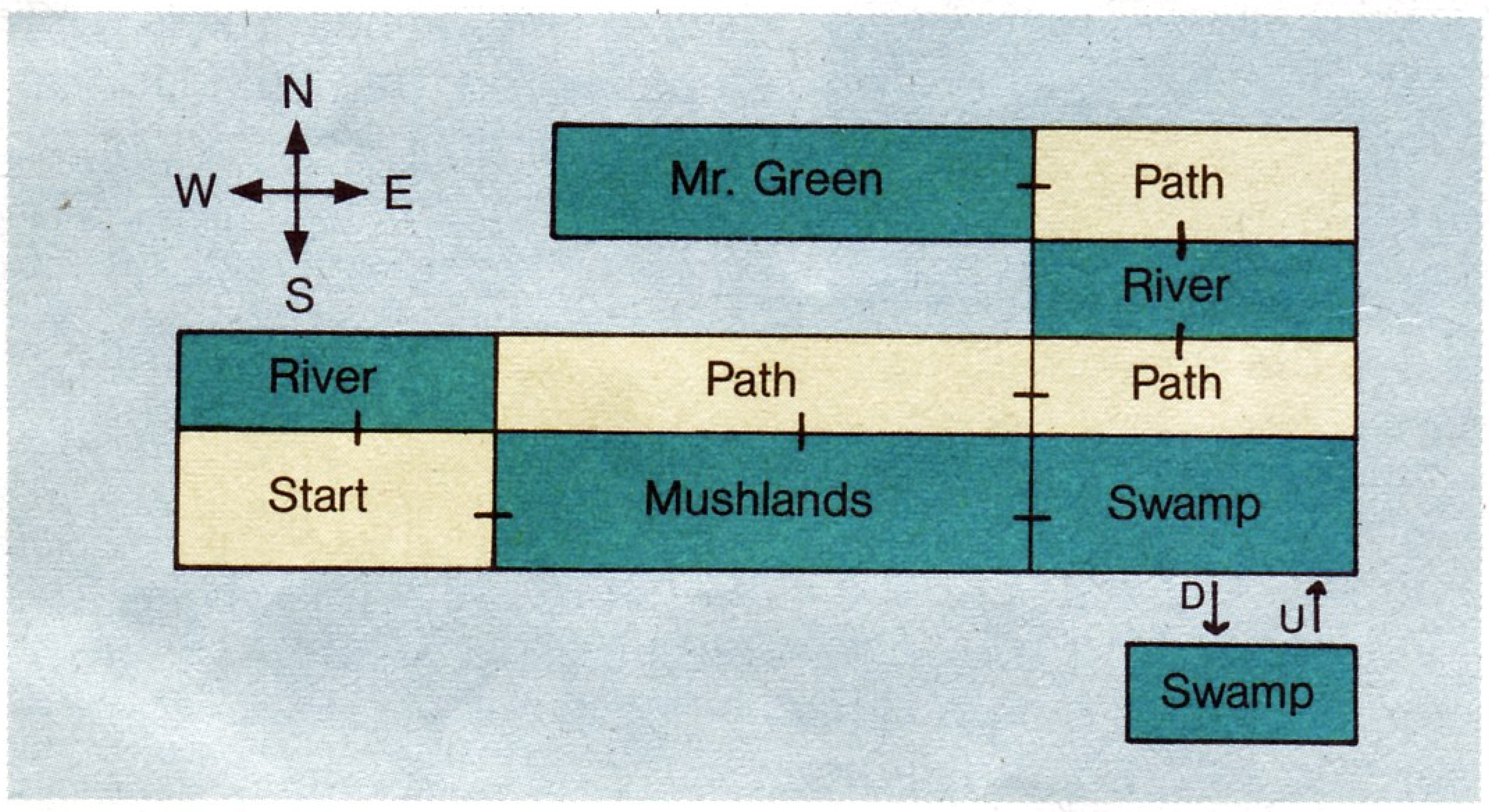


Bild 1. Beispiel eines Lageplans zur Lösung eines Adventure-Spiels

genau so, als hätte es nur einen Zweiwort-Parser zur Verfügung. Das Programm bringt intern einen ganzen Satz sowieso auf die knappste Eingabelänge.

Beachten Sie, daß Gegenstände manipuliert werden oder einen bestimmten Inhalt und spezielle Eigenschaften haben können. Ein Schwert in einem Adventure früherer Jahre leuchtet z.B. blau, wenn Gefahr droht, und rot, wenn der Träger des Schwertes wütend ist. Tatsachen wie diese sollte man auf jeden Fall nutzen, sie helfen enorm bei der Lösung einiger hart zu knackender Nüsse.

Äußerst selten gibt es mehrere Wege, ein Problem zu lösen. Wenn Sie beispielsweise einen Gegenstand an einer Stelle benutzen, an der es auch anders ginge und später einen Punkt erreichen, an dem dieser Gegenstand unbedingt benötigt wird, haben Sie Pech. Das Objekt ist von der vorherigen Benutzung abgenutzt, verbraucht oder kaputt, es kann das »Aus« für Ihr Spiel bedeuten.

Denken Sie logisch darüber nach, was Sie im entsprechenden Raum brauchen und wie Sie es sinnvoll einsetzen. Bei den meisten Abenteuerspielen ist der effektvolle Einsatz eines Gegenstandes nur einmal möglich (abgesehen von ständig nutzbaren Dingen wie Schlüssel oder Kerzen). Wenn erlaubt ist, nur eine bestimmte Zahl von Gegenständen zu tragen, sollten Sie bereits erfolgreich benutzte Dinge wieder ablegen. Speichern Sie lieber vorher den Spielstand ab. Es kann in seltenen Fällen möglich sein, daß Sie den Gegenstand wider Erwarten noch einmal brauchen.

Die meisten Abenteuer haben ihren speziellen »Way of Life«. Das bedeutet: Je nach Softwarehersteller muß man mehr oder weniger logisch denken. Es gibt beispielsweise Adventure-Spiele, bei denen Sie sich zu deren Lösung in die chaotischen Gehirnwindungen eines Verrückten hineindenken und immer das Gegenteil von dem tun müssen, was Sie eigentlich wollten. Bei anderen Abenteuerspielen kommen Sie nur weiter, wenn Sie ein Feingefühl für Einzelsituationen besitzen. Manche Adventures sind extrem logisch aufgebaut, andere dagegen bestehen aus der idealen Mischung von Intuition und Logik.

Spielkarte anlegen

Zu jedem Abenteuerspiel sollten Sie die Räume und ihre Richtungen notieren, noch besser mitzeichnen (Bild 1). Auch wenn Sie die beschriebene Umgebung noch so gut im Gedächtnis haben – irgendwann kommen Sie unweigerlich durcheinander.

Es gibt verschiedene Verfahren, sich Karten und Umgebungspläne zu machen. Haben Sie ein Abenteuer vor sich, das keine schrägen Richtungen wie Südwest oder Nordost besitzt (z.B. in einem Haus mit viereckigen Räumen), ist es am einfachsten und platzsparend, wenn Sie ein kariertes Blatt Papier verwenden. Bezeichnen Sie ein vier Karo großes Quadrat als Raum und setzen Sie die Vierecke direkt nebeneinander. Verbindungen zwischen Räumen kann man mit einfachen Strichen zwischen den Quadraten illustrieren. Beschreibungen des Raumes sind nicht unbedingt für die Karte nötig. Wichtige Einzelheiten und im Raum befindliche Gegenstände müssen innerhalb der Quadrate eingetragen werden. Wenn in den Vierecken nicht ausreichend Platz dafür ist, sollten Sie die Räume nur durchnumerieren und die Gegenstände auf einem anderen Zettel notieren.

Sind Zwischenrichtungen wie Nordost (NO) oder Südwest (SW) im Adventure möglich, sollte man zwischen den einzelnen Quadraten, die die Räume darstellen, etwas Platz lassen. Eventuelle Schrägrichtungen oder Wegbiegungen lassen sich dann besser eintragen. Wie Sie Möbel, Fenster und Türen einzeichnen, bleibt Ihnen überlassen. Das Kartographieren wird in den meisten Fachbüchern zum Thema »Abenteuerspiele« empfohlen.

Eine weitere, wenn auch anspruchsvollere Art des Kartographierens ist die Frei-Hand-Zeichnung. Dazu ist eine Menge künstlerisches Geschick notwendig. Lange Säle können damit als solche dargestellt werden und nicht nur als Vierecke mit langen Verbindungsstrichen. Landschaften lassen sich naturgetreu nachzeichnen. Diese Darstellungsart wird im Lauf der Zeit jedoch sehr unübersichtlich, wenn Sie keine genauen Daten über die Länge einzelner Gänge, Lage der Räume, darin enthaltene Objekte usw. vom Spiel erhalten.

Als idealste Methode für Profi-Abenteuer erweisen sich Mischformen der oben genannten Kartographierungsarten (wie sie auch in manchen Hint Books verwendet werden).

Das Wichtigste: Geduld

Auch wenn der eine oder andere schnell an Abenteuerspielen verzweifelt, üben Sie sich in Geduld. Sterben können Sie in Adventures relativ oft (wir kennen ein Spiel, »Sorcerer«, mit 70 verschiedenen »Todesarten«), aber Leben haben Sie unendlich viele. Fangen Sie einfach wieder von vorn an und vermeiden Sie die Fehler der letzten Spielrunde.

Nutzen Sie alles hier Gesagte aus, auch wenn es noch so unwichtig oder selbstverständlich erscheint. (M. Kohlen/bl)

EINE TEUFLISCHE ERFINDUNG

Spione, Killer und Agenten – Abenteuerspiele dieses Genres zählen zu den beliebtesten. Dieses Text-Adventure wird Sie in Atem halten.

Sie sind Top-Agent des amerikanischen Geheimdienstes SOA (Secret Open Access) und haben eine verantwortungsvolle Aufgabe: Ein verrückter Wissenschaftler hat den »R.A.M.S.« entwickelt, den »**R**e**A**ktor zur Zerstörung der **M**enschlichen **S**eele«. Damit könnte er die gesamte Bevölkerung der Erde ausrotten. Der irre Professor erpreßt mit dieser Höllenmaschine die Regierungen der Welt. Ihm geht es einzig und allein um schnöden Mammon.

Soweit die Vorgeschichte zu diesem Adventure. Wenn Sie sich für fähig halten, den Wahnsinnigen zu stoppen, laden Sie das Programm mit

```
LOAD "R.A.M.S.",8,1
```

Das Spiel startet automatisch. Auf dem Bildschirm blinkt die Meldung "DISK DRIVE A". Wenn nur eine Floppy (Geräteadresse 8) am C64 angeschlossen ist, drücken Sie eine beliebige Taste. Ohne Schnellader dauert es ca. 170 Sekunden, bis der Spaß losgeht. Stören Sie sich nicht daran, daß beim ersten Laden auf dem Schirm »CREATING DISC« erscheint. Damit wird lediglich ein File für die Spielstände (RAMS 0) angelegt, es gehen keine Daten verloren.

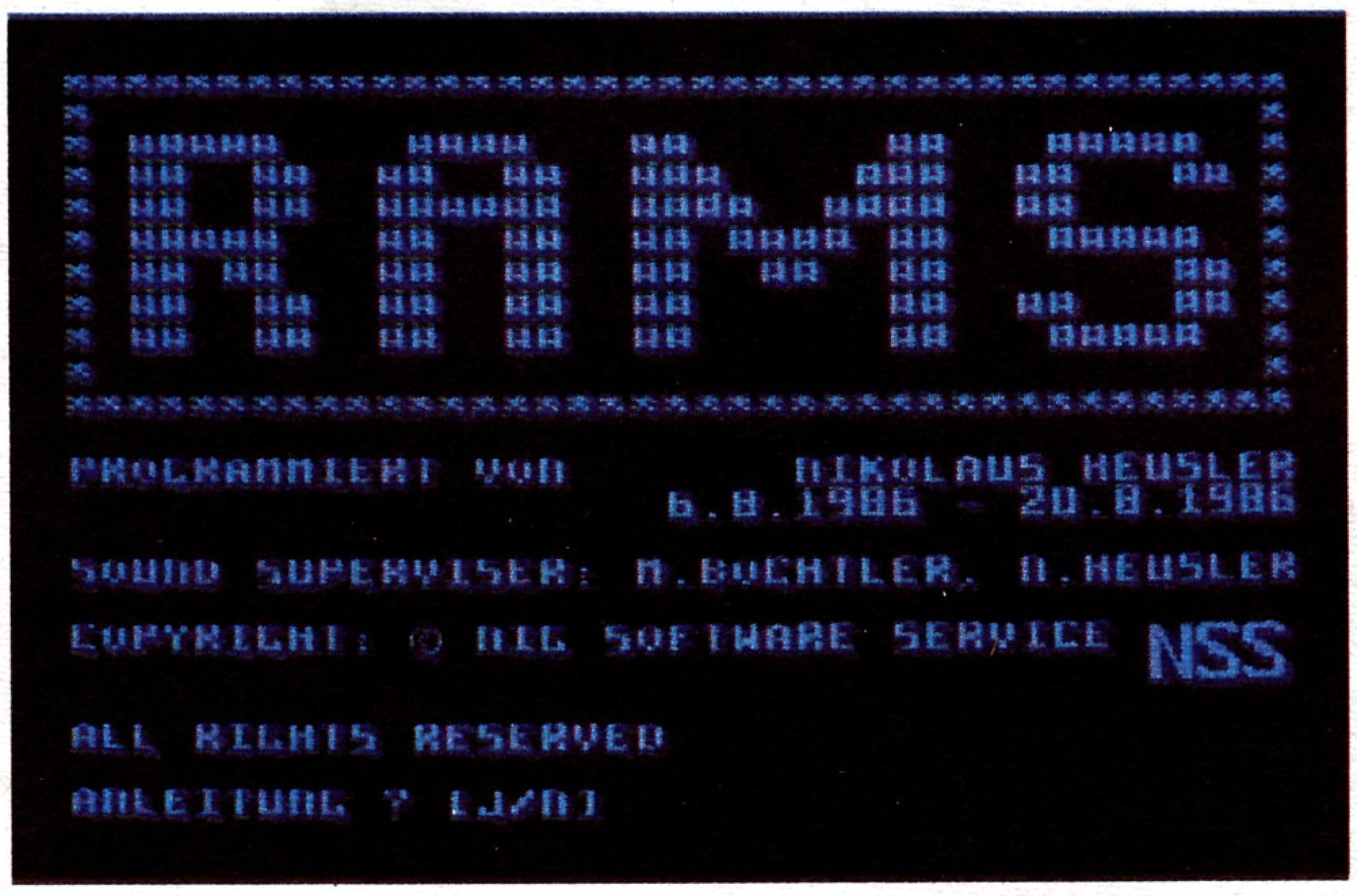


Bild 1. Das Titelbild von »R.A.M.S.«. Die Spielanleitung kann auf Wunsch gezeigt werden.

Befehlseingaben

Der Adventure-Titelscreen (Bild 1) bietet Ihnen an, eine Anleitung zu zeigen, die sich über vier Bildschirme erstreckt und genau über Spielverlauf und Parser (Befehlsinterpretation) Auskunft gibt. Anweisungen werden in zwei Worten eingegeben (Verb und Objekt). Ein Beispiel:

```
SCHAU KAKTUS
```

Es spielt keine Rolle, ob Sie Großschrift, permanent kleine Buchstaben oder beide Schriftarten gemischt verwenden. Das Programm erkennt die Zeichencodes und reagiert darauf. Die Befehle werden in jedem Fall richtig interpretiert. Zwei Anweisungen in einer Zeile können durch »und« getrennt eingegeben werden.

Hier die Erläuterung der wichtigsten Anweisungen:

VOK gibt eine Liste aller Verben aus, die das Spiel versteht (Tabelle 1). Es sind exakt 40 Anweisungen.

NIMM ALLES Wenden Sie diese Befehlsfolge an, wenn sich mehrere Gegenstände in einem Adventure-Raum befinden und Sie alle haben möchten. Die Angabe der Einzelnamen entfällt durch diese Funktion.

SAVE gestattet Ihnen, das Spiel jederzeit zu unterbrechen. Die aktuelle Situation kann abgespeichert werden.

LOAD nach erneutem Laden des Spielstand-Files von Diskette wird exakt an der unterbrochenen Stelle weitergemacht.

CREATE ermöglicht Ihnen, eine separate Datendiskette zur Speicherung des Spielstandes vorzubereiten.

HILFE gibt Tips zur jeweiligen Spielsituation.

RUFE ZENTRALE stellt die Verbindung mit dem Hauptquartier der SOA her.

Das Programm verwendet einen geänderten Zeichensatz, alle Textausgaben erscheinen formatiert im Blocksatz.

»R.A.M.S.« gehört keinesfalls zu den leichten Adventures. Bereits zu Beginn des Spiels kann es passieren, daß tiefe Dunkelheit Sie einhüllt. Keine Richtung wird mehr akzeptiert. Hier bleibt Ihnen nichts anderes übrig, als sich durch die Anweisung »RUFE ZENTRALE« mit dem Agenten-Hauptquartier in Verbindung zu setzen. Wissen Sie allerdings Ihr Paßwort nicht, haben Sie keinen Erfolg. Daher geben wir Ihnen den ersten (und letzen) Tip zu diesem Adventure: Das Kennwort steht auf einem Zettel in einem Umschlag. Wo der versteckt ist, müssen Sie selbst herausfinden.

Verzweifeln Sie nicht, wenn Sie nicht einmal mehr der »HILFE«-Befehl weiterbringt: Probieren Sie alle erdenklichen Möglichkeiten aus. Auch bei total unsinnigen Eingaben stürzt das Programm nie ab. Die Lösung ist in mindestens 72 Spielzügen zu schaffen.

Ärgern Sie sich nicht, wenn es beim ersten Anlauf nicht sofort klappt. Beim nächsten Versuch haben Sie mehr Glück!

Haben Sie nach langen Irrwegen endlich das Ziel erreicht, fragt Sie das Programm, ob es eine Zeitungsmeldung ausdrucken soll. Die Routine wurde für die Drucker MPS 801/803 geschrieben, doch die meisten anderen Drucker können ebenso problemlos verwendet werden.

Viel Spaß bei der Lösung von »R.A.M.S« – hoffentlich retten Sie die Menschheit!

(Nikolaus Heusler/bl)

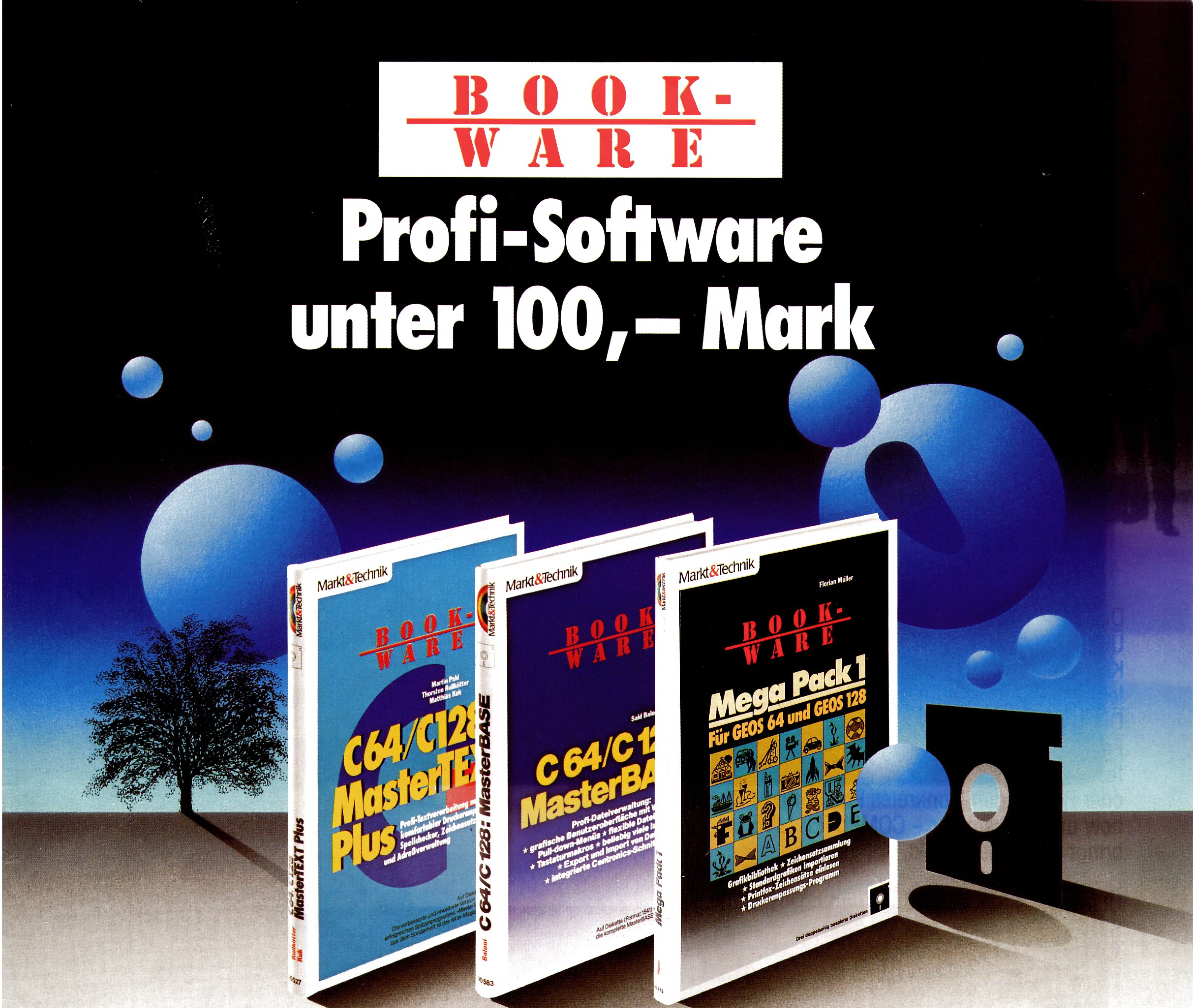
Befehlsübersicht »R.A.M.S.«

schau	nimm	lege	nehm
Vokabular	Inventur	save	load
Ende	speichere	hilfe	Zeit
create	rufe	oeffne	schliesse
lese	warte	springe	gehe
betrete	klettere	lies	hebe
benutze	bewege	zerstoere	starte
gib	gebe	wirf	werfe
schiebe	druecke	betaste	stecke
ziehe	mache	schichte	drehe

Tabelle 1. Diese Befehle versteht das Text-Adventure

Kurzinfo: R.A.M.S.

Programmart: Abenteuerspiel
Spielziel: Ein verrückter Wissenschaftler bedroht die Welt. Machen Sie ihn unschädlich.
Laden: LOAD "R.A.M.S.",8,1
Starten: Programm startet automatisch
Steuerung: Tastatur
Benötigte Blocks: 146 Blocks
Programmautor: Nikolaus Heusler



M. Pahl, T. Rullkötter, M. Kuk
C64/C128 MasterText Plus
Die leistungsfähige Textverarbeitung: jetzt mit Rechtschreibkorrektur und Adreßverwaltung.
1988, 201 Seiten, inkl. Programmdiskette
ISBN 3-89090-527-7
DM 59,–* (sFr 54,30*/öS 502,–*)

F. Müller
Mega Pack 1 für GEOS 64 und GEOS 128
250 Kleingrafiken, 190 Zeichensätze, 2 Konvertierungsprogramme und ein Druckprogramm.
1989, 160 Seiten,
inkl. 3 Programmdisketten
ISBN 3-89090-772-5
DM 59,–* (sFr 54,30*/öS 502,–*)

S. Baloui
C64/C128 MasterBase
Die professionelle Dateiverwaltung.
1988, 155 Seiten, inkl. Programmdiskette
ISBN 3-89090-583-8
DM 59,–* (sFr 54,30*/öS 502,–*)

S. Vilsmeier
3-D-Konstruktion mit Giga-CAD Plus
Die überaus positive Resonanz aller Leser war der Anlaß, Giga-CAD für den C64/C128 in einer verbesserten Version vorzustellen.
1986, 183 Seiten, inkl. 2 Programmdisketten
ISBN 3-89090-409-2
DM 49,–* (sFr 45,10*/öS 417,–*)

W. Oppacher, K. Oppacher, M. Wenzel
C64/C128 Giga-Paint
Ein professionelles Mal- und Zeichenprogramm.
1988, 261 Seiten, inkl. 2 Programmdisketten
ISBN 3-89090-619-2
DM 59,–* (sFr 54,30*/öS 502,–*)

S. Vilsmeier
C64/C128 Objekt-Bibliotheken zu Giga-CAD Plus
Eine Sammlung von neuen Objekten, Zeichensätzen und Utilities für Giga-CAD Plus.
1988, 64 Seiten, inkl. 2 Programmdisketten
ISBN 3-89090-581-1
DM 39,–* (sFr 35,90*/öS 332,–*)

W. Oppacher, K. Oppacher, M. Wenzel
C64/C128 Tools für Giga-Paint
Eine Sammlung von Erweiterungen für Giga-Paint, die von einfachen Utilities (z.B. Maustreibern) bis zu sehr vielseitigen Modulen zur Grafiknachbearbeitung reichen (z.B. Bilder in beliebige Formen pressen).
1989, 296 Seiten, inkl. 2 Programmdisketten
ISBN 3-89090-138-7
DM 59,–* (sFr 54,30*/öS 502,–*)

C. Clasohm
GeoTerm
Mit GeoTerm erhalten Sie ein professionelles Terminalprogramm mit grafischer Benutzeroberfläche unter GEOS.
1989, 107 Seiten, inkl. Programmdiskette
ISBN 3-89090-757-1
DM 69,–* (sFr 63,50*/öS 587,–*)

* Unverbindliche Preisempfehlung

In Vorbereitung

F. Müller
Mega Pack 2 für GEOS 64 und GEOS 128
Lieferbar 4. Quartal 1989,
ca. 150 Seiten, inkl. Diskette
ISBN 3-89090-350-9
ca. **DM 59,–***

INFO-COUPON

Bitte senden Sie mir Ihr Gesamtverzeichnis
mit über 500 aktuellen Computerbüchern und Software

Name ____________________

Straße ____________________

PLZ/Ort ____________________

Bitte ausschneiden und einsenden an: Markt&Technik Verlag AG,
Buch- und Software-Verlag, Hans-Pinsel-Straße 2, 8013 Haar 64 SH 52

In Vorbereitung

W. Knupe/H.-J. Ciprina/R. Bonse/
V. Goehrke
MegaAssembler
Lieferbar 4. Quartal 1989,
ca. 400 Seiten, inkl. Diskette
ISBN 3-89090-247-2
ca. **DM 89,–***

Markt&Technik-Bücher und -Software erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

132/910